Variable-Length Error-Correcting Codes

A thesis submitted to the University of Manchester

for the degree of Doctor of Philosophy in the Faculty of Science

1995

Victor Buttigieg

Department of Electrical Engineering

Contents

APPENDIX B

APPENDIX C

List of Figures

List of Tables

Abstract

Variable-length error-correcting (VLEC) codes are considered for combined source and channel coding. Instantaneous decoding algorithms for VLEC codes treated previously in the literature are found to suffer from loss of synchronisation over the binary symmetric channel, consequently resulting in poor performance. A novel maximum likelihood decoding algorithm, based on a modified form of the Viterbi algorithm, is derived for these codes by considering the spatial memory due to their variable-length nature. This decoding algorithm achieves a large coding gain (from 1 to 3 dB) over the instantaneous algorithms because of its good synchronisation properties.

The performance of these codes with maximum likelihood decoding when compared to standard cascaded source and channel coding schemes with similar parameters is found to be slightly better (about 0.5dB gain). However, the decoding complexity for VLEC codes is greater. This problem is solved by implementing a sequential decoding strategy, which, for almost the same performance, offers a much reduced computational effort (about an order of magnitude less) when the signal-to-noise ratio on the channel is relatively high.

The synchronisation performance of VLEC codes with maximum likelihood decoding over channels which admit symbol deletion or insertion errors is also found to be good (synchronisation is recovered within less than two source symbols following an error).

Various properties of VLEC codes influencing their performance both with maximum likelihood and sequential decoding are defined and characterised. A union bound on their performance over the binary symmetric channel is derived. Several different constructions for VLEC codes are given, one of which optimises the average codeword length for a given source while attaining the required error-correcting power.

Declaration

No portion of the work referred to in the thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

Copyright Notice

About Author

Victor Buttigieg received the B.Elec.Eng. (Hons) degree in *Electrical Engineering* from the University of Malta in 1990. For a brief period he taught at the Fellenberg Technical Institute for Industrial Electronics, Malta, before joining the Faculty of Electrical and Mechanical Engineering at the University of Malta as an Assistant Lecturer. In 1991 he was awarded a three-year Commonwealth Academic Staff Scholarship by the Association of Commonwealth Universities at the University of Manchester. He was awarded the degree of M.Sc. in *Digital Systems Engineering* from the University of Manchester in 1992.

Acknowledgements

First of all I wish to thank Professor P.G. Farrell, my supervisor, for the excellent way in which he guided me throughout this project. Without his constant encouragement, sense of humour and many hours of stimulating discussions this work would not have reached its present form.

To all fellow students and staff in the Communications Research Group at the University of Manchester goes my sincere thank you. The many interesting discussions, technical and non-technical, and friendship made life in Manchester that much easier. My special thanks goes to Jon Larrea, who for many months was my dependable link with Manchester.

I also wish to thank the staff of the Department of Communications and Computer Engineering at the University of Malta, for giving me time to complete this thesis during the last few months back at my department.

I also acknowledge the financial support of the Association of Commonwealth Universities and the British Council, through the award of a Commonwealth Academic Staff Scholarship.

Last but not least I want to thank my wife, Rose Marie, and my son, Darren, for all the time, love and support they give me, without which this thesis would not have been possible.

List of Abbreviations

ASCII	American Standard Code for Information Interchange
AWGN	Additive White Gaussian Noise
BCH	Bose-Chaudhuri-Hocquenghem
BMS	Binary Memory-less Source
BPSK	Binary Phase Shift Keying
BSC	Binary Symmetric Channel
BSD	Bounded Synchronisation Delay
CAS	Compare Add Select
CDF	Column Distance Function
FSVLEC	Finite State Variable-Length Error-Correcting
GA	Greedy Algorithm
GCD	Greatest Common Divisor
HDLC	High-level Data Link Control
iff	If and only if
LHS	Left Hand Side
MAP	Maximum A-Posteriori
MLD	Maximum Likelihood Decoding
MVA	Majority Voting Algorithm
NASA	National Aeronautical and Space Agency
RHS	Right Hand Side
RS	Reed-Solomon
SEP	Symbol Error Probability
SSVLEC	State Splitting Variable-Length Error-Correcting
VLEC	Variable-Length Error-Correcting
VLSI	Very Large Scale Integration

Glossary of Symbols

$\forall$	For all
α	Admissibility (or Error) mapping
χ	Index of comma freedom
δ	Maximum distance for an anticode
ϕ	Forced transition without any input symbols
ϕ_{MAP}	MAP factor
η	Code efficiency
η_m	Total number of source symbols in message m
κ	Positive constant
λ	Empty word
μ	The minimum number of consecutive states in the repetitive part of the trellis diagram for a VLEC code C required to build up all subsequent states
θ	Positive constant
σ	Number of different codeword lengths
ν	Average number of information bits required to encode a source symbol
ξ	Average number of paths which visit the top of the stack per transmitted source symbol
$\mathcal{A}$	Fixed-length anticode
A	Information source
A_h	The average number of converging pairs of paths at Hamming distance h
$\mathbf{a}$	Sequence of source symbols
a	Source symbol
$\mathcal{a}, \mathcal{b}, \mathcal{c}, \mathcal{d}$	Codewords of $\mathcal{F}$
B_h	Average Levenshtein distance between all converging pairs of paths whose encoded messages are at a Hamming distance h from each other
b_k	Minimum block distance for codewords of length L_k
$b_{\min}$	Overall minimum block distance
C	VLEC code
$C(\mathbf{a}, \mathbf{b})$	Converging distance between **a** and **b**
C_h	Average number of source symbols in all converging pairs of paths whose encoded messages are at a Hamming distance h from each other
c	Code symbol
$\mathbf{c}_i$	A codeword of code C
$c_{\min}$	Minimum converging distance
D	The number of states in the decoding window
$D(\mathbf{a}, \mathbf{b})$	Diverging distance between **a** and **b**
d	Minimum distance for a block code
$d_c(\eta)$	CDF

d_{free}	Free distance
$d_{\min}$	Minimum diverging distance
d_u	Unequal length free distance
E	Expanded code for VLEC code under error mapping α
$E_{s_{\text{eff}}}$	Average effective error span
E_s	Average error span
E_x	Average number of extra paths extended per source symbol more than the minimum number
e	Number of errors
$\mathcal{F}$	Fixed-length code
$F(\mathbf{u}_m, \mathbf{y})$	Fano metric for message $\mathbf{u}_m$ given received sequence $\mathbf{y}$
F_N	Extended code of order N for VLEC code C
$\mathbf{f}_i$	A codeword of the extended code F_N
f_N	Cardinality of F_N
$G_{q,r}$	The set of all pairs of path segment indices corresponding to path segments which diverge at state S_q and merge again for the first time at state S_r
g	The GCD of the codeword lengths of C
$H(A)$	Entropy of source A
$H(\mathbf{a},\mathbf{b})$	Hamming distance between $\mathbf{a}$ and $\mathbf{b}$
H_i	Horizontal sub-code
h	Hamming distance between two given codewords
K	Constraint length
$\mathbf{k}$	Information vector
k	Number of information bits
L	Codeword length for fixed-length code
$L(\mathbf{a}, \mathbf{b})$	Levenshtein distance between $\mathbf{a}$ and $\mathbf{b}$
L_{average}	Average codeword length for a given code and source
L_i	The ith different codeword length
l_i	Length of codeword $\mathbf{c}_i$
M_i	The metric value for state S_i
m	Block length for anticode
m_j	The metric for codeword $\mathbf{c}_j$
N	Number of bits in the encoded message
N_{m}	Number of bits in encoded message m
N_S	The total number of states in the trellis
n	Block length
n	Number of bits lost at the start of the message
n_i	Number of codewords in a given path with length L_i
p	Cross-over probability for the BSC
$\mathbf{p}$	Proper prefix of a word
$P(a)$	Probability of occurrence of a
$P(E)$	Error event probability
$P(E, r)$	The error event probability at bit position r.
P_0	The probability measure induced on the channel output alphabet when the channel inputs are used according to some probability distribution $Q(\,)$
$P_f(E)$	The first error event probability at any bit position
$P_f(E, r)$	The first error event probability at bit position r
P_h	Probability of decoding a sequence into another sequence at distance h over the BSC

P_m	Probability of message *m*.
p_m	Transmitted path through tree
P_N	The set of paths through trellis (or tree) of length *N* bits
$P_s(E)$	Symbol error probability
p_i	The *i*th path through tree
p_N^m	Minimum distance path to state S_N
$\left(p_{q,r}^i\right)_{\beta_i}$	The first β_i branches of the path segment $p_{q,r}^i$
p_r^i	The *i*th path through trellis going to state S_r
$p_{q,r}^i$	The segment of the path p_r^i from state S_q to state S_r
$Q(\,)$	Probability distribution for channel input alphabet
Q_i	Segment decomposition of a VLEC code
$\mathbf{q}$	Codeword segment
q	Number of code symbols
q, r	Bit position
R	Code rate
S_i	State in trellis diagram representing bit position *i*
s	Number of codewords/source symbols
$\mathbf{s}$	Proper suffix of a word
s_i	Number of codewords with length L_i
$\tilde{s}_i$	Number of codewords with length less than L_i
T	Maximum decoding delay
t	Number of correctable errors per segment or per codeword
$\mathbf{u}_m$	Codeword sequence corresponding to message *m*
$\mathbf{v}$	State label for a FSVLEC code
V_i	Vertical sub-code
W	Allowed set of words satisfying given conditions
W	Decoding window depth in bits
$W(\mathbf{a})$	Hamming weight for **a**
$\mathbf{w}, \mathbf{x}, \mathbf{y}, \mathbf{z}$	Words over X
$\mathbf{w}_d$	Received word
X	Code alphabet
x_i	Code symbol
$\mathbf{y}$	Received bit sequence
Z	Maximum synchronisable delay
z	Synchronisable delay

To Rose Marie and little Darren

Chapter 1.

Introduction

1.1. Introduction

The aim of any communication system is to transmit information from some source at point A, to some sink at point B over some channel. The channel can take several forms, such as a physical cable, a wireless link or even a storage device. The communication system is successful if it transmits this information faithfully and efficiently. If the information source is digital in nature, such as computer data for instance, then we may require that the reproduction of the information at the sink will be an exact replica of the source data. In other instances, especially for analogue information sources, such as speech, we allow some distortion at the sink. This distortion may result from noise on the communication channel or even from the way the source is transmitted. For instance, transmitting a speech signal over a digital communication system will always introduce some distortion, even for a noiseless channel, since in order to digitise the signal a finite number of quantisation levels must be used. Currently, most communication systems are being implemented using digital technology due to a host of advantages, the most important of which are its ease of implementation using VLSI technology and its superior performance in noise. Digital transmission also allows a host of signal processing techniques which would otherwise be impossible or difficult to implement in analogue form. This thesis treats one such technique, to perform combined source and error-correction coding.

Figure 1.1 shows the basic block diagram for a digital communication system, where the source is already assumed to be in digital form [Viterbi & Omura, 1979]. The source encoder is used to remove as much as possible the redundancy present in most natural information sources, performing what is commonly known as data compression. The source encoding could either be a one-to-one mapping, in which case the source may be reproduced exactly if the source coded data is transmitted over an error free channel, or it could be a many-to-one mapping. In this latter case, although better compression may be achieved, the source can only be recovered within some fidelity criterion. In this thesis we are always going to assume that the source is discrete, memory-less and stationary, i.e. the probability of occurrence of any source symbol is independent of previously emitted symbols and independent of time. Also, we are only going to consider distortionless source coding.

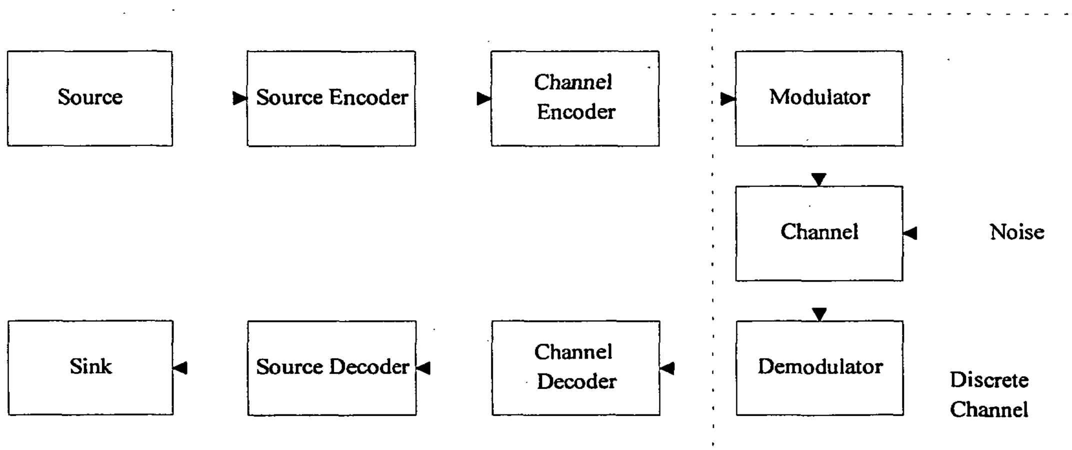

Figure 1.1: Standard basic digital communication system

In practice, noise is usually present, in one form or another, on any communication channel. In a digital system this translates into errors in the received sequence of symbols. There are several techniques one could adopt in order to reduce these errors, such as increasing the signal power, reducing the transmission speed, and so on. One of the techniques which has been gaining ground over the last several years is that of error-

correction, whereby the source coded data is further encoded using a so-called error-correction code [Lin & Costello, 1983]. The channel encoder may involve other levels of coding (such as run-length limited encoding for magnetic recording, for instance [Schouhamer Immink, 1990]). Here, however, we shall use error-correction and channel coding as synonymous. Error correction is achieved by introducing structured redundancy in the data. Through this redundancy, the decoder can determine that errors have occurred during a transmission (error-detection) or even more powerful, which of the transmitted symbols are in error. Even from this simplistic overview, the dual nature for source and channel coding is already apparent, whereby source coding is removing redundancy for efficient transmission and channel coding is re-introducing redundancy to combat errors on the channel. This duality is in fact much deeper, as established by Shannon's source and channel coding theorems [Shannon, 1948].

1.2. Combined Source and Channel Coding

Since source coding is removing redundancy and channel coding is reintroducing it, albeit in a different form, we may query if it is better to combine these two operations into a single operation, as shown in Figure 1.2. However, a direct consequence of Shannon's work is precisely that these two operations may be separated without any loss in performance, for most common sources and channels. This has become known as the separation theorem. Note, however, that the separation theorem does not hold for certain classes of sources and/or channels [Vembu *et al.*, 1995].

Separating the two operations has the advantage that if the source is changed in a system, the only component that needs to be modified is the source encoder/decoder pair. Similarly, if the characteristics of the channel change, then it is only the channel encoder/decoder pair that needs to be replaced. Shannon's work gives little insight, however, into how complex the system may become by separating the two operations. His work does not preclude the possibility that by combining the two, the overall system complexity for a given performance may be reduced. Massey [1978] has investigated this problem for the special case of combined linear source and channel coding applied to a

binary memoryless source (BMS) and a binary symmetric channel (BSC). He has found that for the distortionless case, a combined source and channel linear encoder is simpler to implement and is as optimal as separate encoders. Interestingly enough, this result does not hold when some distortion is allowed at the decoder, where here the combined scheme would be sub-optimal. Obviously, once the two encoders are combined, there is the disadvantage that the system becomes less flexible, in that any change in the source and/or channel statistics will entail a change in the combined encoder (and decoder).

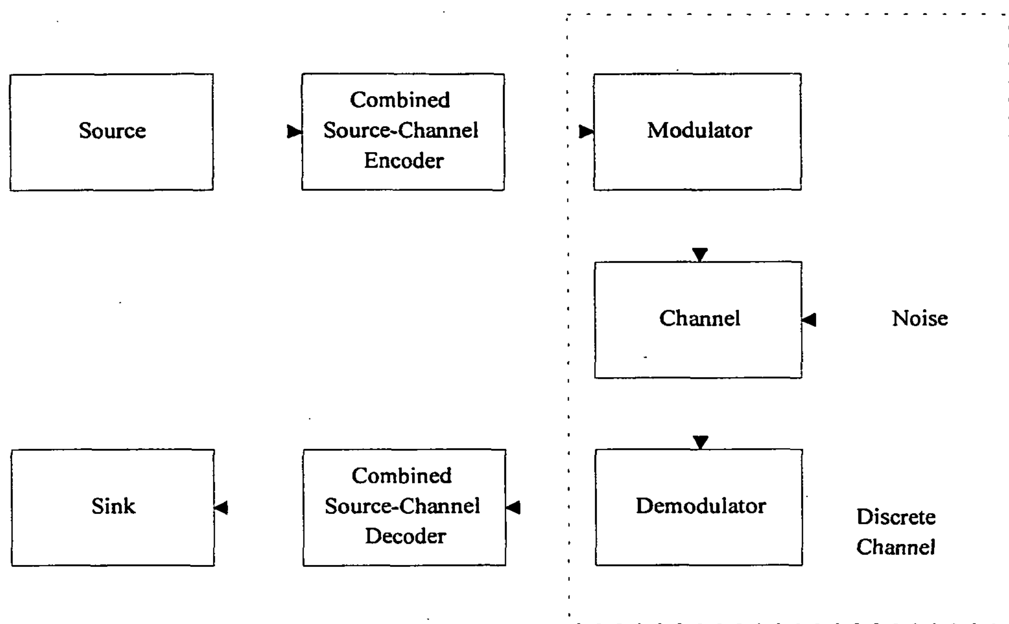

Figure 1.2: Combined source and channel coding

1.3. Variable-Length Error-Correcting Codes

Error-correcting codes can be broadly classified as block or convolutional [Lin & Costello, 1983]. The main difference between the two lies in the fact that in block codes, k symbols of information are mapped into n code symbols, not necessarily from the same alphabet, where $n > k$. In convolutional codes, a similar mapping takes place, but in this case it also depends on previous inputs. Hence, convolutional codes have memory. The

amount of memory, or constraint length [Viterbi, 1971], will determine the performance. For this reason, in convolutional codes n and k are usually very small: $k = 1$ and $n = 2$ are normal for these codes, whereas the corresponding values in the case of block codes are of the order of a few hundreds, since here it is the block length which determines the performance[1]. For instance, an $n = 255$, $k = 223$, denoted by (255,223), Reed-Solomon (RS) code with 8-bit symbols is a standard block code used in deep space communication by NASA [Sweeney, 1991].

In this thesis we examine a new class of error-correcting codes which we shall call VLEC (Variable-Length Error-Correcting) codes. As the name implies, the main difference between these codes and the standard block and convolutional codes is the fact that the codewords are of variable length. The codes that we investigate here are similar to block codes in the respect that each codeword is mapped to a given set of information symbols irrespective of the previous inputs. However, their main characteristics are very similar to those of convolutional codes. This similarity is brought about by the fact that the position of any codeword within the encoded message depends on the previously occurring codewords and hence VLEC codes exhibit a form of "spatial memory". In the case of convolutional codes the memory in the encoder directly affects the value of the output, but not its position.

In addition, due to their variable length nature, VLEC codes may be used to perform combined source and channel coding by assigning the shorter codewords to the more probable source symbols. In the encodings considered in this thesis, each source symbol is mapped to a single codeword according to the source statistics. However this could easily be extended to have multiple source symbols mapped to single codewords in order to increase the code efficiency by increasing the number of codewords in the VLEC code. The number of source symbols mapped to each codeword may not necessarily be fixed, resulting in variable-to-variable length encoding.

[1] Since a large block length averages out the effects of noise and enables good codes to be constructed.

1.4. Thesis Structure

The first published work on VLEC codes that we are aware of is that edited by Hartnett [1974], who compiled a series of reports originating at Parke Mathematical Laboratories, Massachusetts, U.S.A between 1957 and 1968. Since then not much has appeared apart from recent work by Dunscombe [1988], Bernard and Sharma [1990] and Escott [1995]. This work is treated in Chapter 2, which also includes a general exposition of variable-length codes. However, all the previous work on VLEC codes has completely ignored the spatial memory inherent in VLEC codes. Consequently, the performance of these codes with the published decoding algorithms is not very good, which may partly explain why they are not treated much in the literature. The spatial memory of VLEC codes was first considered by Buttigieg [1992]. This work is improved upon in Chapter 3, where a maximum likelihood decoding algorithm for these codes is derived. This algorithm offers substantial coding gain over the earlier decoding algorithms for VLEC codes, but is achieved at the price of increased complexity. This drawback is tackled in Chapter 4, where a sequential decoding algorithm based on the stack algorithm for convolutional codes is given. It is shown that for reasonably large signal to noise ratios, the decoding complexity is greatly reduced over the maximum likelihood decoding algorithm, while maintaining the same performance.

One of the main problems with variable-length codes in general is that of loss of synchronisation. In our opinion this is another reason why VLEC codes were not considered much in the literature. The main objective of error-correcting codes is to reduce the effect of errors on the channel. Loss of synchronisation has the opposite effect whereby errors on the channel may be propagated by the decoder. This is especially evident with the decoding algorithms found in the literature. In Chapter 5 we show that VLEC codes with maximum likelihood decoding have reasonably good synchronisation properties over the BSC and will also perform well over channels which allow deletion or insertion of channel symbols. These kinds of errors are especially problematic in the case of standard error-correcting codes due to their fixed-length nature.

Having determined which properties of VLEC codes influence their performance in the earlier chapters, Chapter 6 discusses issues involved with their construction and gives two construction algorithms. Codes for the 26-symbol English source and the 128-symbol ASCII source are constructed and their performance compared with standard error-correcting codes with and without source coding.

Finally, in Chapter 7 we draw some conclusions on the performance of VLEC codes for combined source and channel coding. We also give some new ideas to improve their performance and list some open problems.

Chapter 2.

Variable-Length Error-Correcting Codes

2.1. Introduction

Variable-length codes are normally used for source coding. Consequently, they are frequently considered in conjunction with noiseless channels. Hence, we will first review some properties that characterise variable-length codes for the noiseless case.

The problem of synchronisation is then considered, both in the general case and in particular for variable-length codes. This is the main problem area for variable-length codes, which limits their use in practice. We will then consider variable-length codes capable of correcting substitution errors. In particular, the special class of α-prompt codes is considered in detail, and three instantaneous decoding algorithms for these codes are given.

Most of the work presented in this chapter has appeared previously in the literature, as will be indicated. However, there are a few extensions of previous work in Sections 2.6, 2.8 and 2.9. In particular, in Section 2.9.2, a practical algorithm to determine the symbol error probability in the case of variable-length codes is given. This is suitable for use in computer simulations to determine the performance of variable-length codes.

2.2. Variable-Length Codes

Let X be a code alphabet with cardinality q. A finite sequence $\mathbf{w} = x_1x_2\cdots x_l$ of code symbols is called a word over X of length $|\mathbf{w}| = l$, where $x_i \in X$, for all $i = 1, 2, \cdots, l$. Denote the set of all finite-length words over X by X^+. Note that if λ denotes the empty

word, $\lambda \notin X^+$. Let $X^* = X^+ \cup \lambda$. Given **w**, **p**, **s** $\in X^+$, if **w** = **ps**, then **p** is a proper prefix of **w** and **s** is a proper suffix of **w**. A set C of words is called a code. Note that $C \subset X^+$. Similarly, denote the set of all finite-length sequences of codewords of C by C^+ and let $C^* = C^+ \cup \lambda$.

Let the code C have s codewords $\{\mathbf{c}_1, \mathbf{c}_2, \cdots, \mathbf{c}_s\}$ and let $l_i = |\mathbf{c}_i|$, $i = 1, 2, \cdots, s$. Without loss of generality, assume that $l_1 \le l_2 \le \cdots \le l_s$. Further, let σ denote the number of different codeword lengths in the code C and let these lengths be $L_1, L_2, \cdots, L_\sigma$, where $L_1 < L_2 < \cdots < L_\sigma$. Let the number of codewords with length L_i be s_i, and the number of codewords with length less than L_i be $\tilde{s}_i$, i.e. $\tilde{s}_i = \sum_{j=1}^{i-1} s_j$. Note that $\tilde{s}_1 = 0$ and that $L_1 = l_{\tilde{s}_1+1} = l_1$, $L_2 = l_{\tilde{s}_2+1}$, $\cdots$, $L_\sigma = l_{\tilde{s}_\sigma+1} = l_s$ and $s = \sum_{i=1}^{\sigma} s_i$. We shall use $(s_1@L_1, s_2@L_2, \cdots, s_\sigma@L_\sigma)$ to denote such a code. We shall later expand this notation for the case of variable-length error-correcting (VLEC) codes (c.f. Chapter 3).

If $\sigma = 1$, then C is a fixed-length code. Hence, we shall define a variable-length code C to be a code with $\sigma > 1$. Further, if $q = 2$, then the code will be binary and X could be taken to be the set $\{0, 1\}$. Unless otherwise stated, it will be assumed throughout this thesis that C is a binary variable-length code. However, most of the results obtained may easily be extended to non-binary codes.

The *Hamming weight* (or simply *weight*) of a word **w**, $W(\mathbf{w})$, is the number of non-zero symbols in **w**. The *Hamming distance* (or simply *distance*) between two equal length words, $H(\mathbf{w}_1,\mathbf{w}_2)$, is the number of positions in which $\mathbf{w}_1$ and $\mathbf{w}_2$ differ. For the binary case, it is easy to see that $H(\mathbf{w}_1,\mathbf{w}_2) = W(\mathbf{w}_1+\mathbf{w}_2)$, where the addition is modulo-2.

Let A be a memory-less data source with s source symbols $\{a_1, a_2, \cdots, a_s\}$, each with probability of occurrence $P(a_i)$, $i = 1, 2, \cdots, s$, with $\sum_{i=1}^{s} P(a_i) = 1$. Without loss of generality, assume that $P(a_1) \ge P(a_2) \ge \cdots \ge P(a_s)$. The source A is encoded using code C by mapping symbol a_i to codeword $\mathbf{c}_i$ for all $i = 1, 2, \cdots, s$. It is easy to prove that this mapping is the most efficient given code C and source A. In this case, the average codeword length is given by

$$L_{\text{average}} = \sum_{i=1}^{s} l_i P(a_i) . \tag{2.1}$$

2.3. Some Properties of Variable-Length Codes

If we compare variable-length codes to fixed-length codes, we find that the former are much more difficult to deal with, since in this case there is also a degree of ambiguity in determining the codeword boundaries. In the case of fixed-length codes, once we know where a codeword starts, then from that point onwards it is very easy to determine the subsequent boundaries, assuming that the channel does not insert or delete code symbols. In the previous statement, there are two important "ifs", which when not satisfied will give big problems in the case of fixed-length codes. We shall comment further on this in Section 2.4.

2.3.1. Non-Singular Codes

A code C is said to be *non-singular* if all the codewords in the code are distinct [Abramson, 1963]. Both fixed and variable-length codes must satisfy this property in order to be useful. This property is trivial to check.

2.3.2. Unique Decodability

A code C is said to be *uniquely decodable* if we can map a string of codewords unambiguously back to the correct source symbols. It is obvious that all fixed-length codes which are non-singular are uniquely decodable. However, this is not in general true for variable-length codes. We will show this with an example. Consider the code {0, 01, 10} used to encode the source $\{a, b, c\}$. Clearly this is a non-singular code since all codewords are distinct. However, the message *ac* which is encoded as 010, cannot be uniquely decoded since the codeword sequence 010 may either be decoded as *ac* or as *ba*. Necessary and sufficient conditions for unique decodability and an algorithm to test these conditions are given by Sardinas and Patterson [1953]. Hazeltine [1963] gives an alternative algorithm to determine if a code is uniquely decodable.

A uniquely decodable code C has *finite decoding delay* T iff there exists an integer T such that if $\mathbf{x} \in X^+$, $|\mathbf{x}| \geq T$, and $\mathbf{xy} \in C^+$, then $\mathbf{x}$ has a decomposition $\mathbf{x} = \mathbf{x}_1\mathbf{x}_2$ such that whenever $\mathbf{xz} \in C^+$ then $\mathbf{x}_1 \in C$ and $\mathbf{x}_2\mathbf{z} \in C^*$; in words, iff the first T code symbols in a message are sufficient to determine the first codeword. A variable-length code may, for

certain messages, exhibit an infinite decoding delay and hence will not be suitable for practical use. Several people have worked on the determination of the decoding delay for variable-length codes. The first to give an algorithm to calculate this was Even [1963].

A necessary and sufficient condition for the existence of a uniquely decodable code is provided by the McMillan inequality [McMillan, 1956].

Theorem 2.1: A necessary and sufficient condition for the existence of a uniquely decodable code with codeword lengths $l_1, l_2, \cdots, l_s$ is that

$$\sum_{i=1}^{s} q^{-l_i} \leq 1 \tag{2.2}$$

where q is the number of different code symbols.

Proof: [Abramson, 1963] The sufficient part is proved by construction. From expression (2.2) we obtain a series of inequalities on the number of codewords, s_i', of a given length i

$$s_{L_\sigma}' \leq q^{L_\sigma} - s_1' q^{L_\sigma - 1} - s_2' q^{L_\sigma - 2} - \cdots - s_{L_\sigma - 1}' q \tag{2.3}$$

$$\cdots$$

$$s_3' \leq q^3 - s_1' q^2 - s_2' q \tag{2.4}$$

$$s_2' \leq q^2 - s_1' q \tag{2.5}$$

$$s_1' \leq q \tag{2.6}$$

Assuming the codeword lengths satisfy (2.2) and using expressions (2.3)-(2.6) we may construct a uniquely decodable code as follows. We require $s_1' \leq q$ codewords of length one. Since there are q code symbols then we may choose any arbitrary set of s_1' unique code symbols as codewords of length one. One way of ensuring that the code be uniquely decodable is to enforce that the remaining codewords start with different code symbols, thus creating a prefix code (see the next section). Hence there is the possibility of forming $(q-s_1')q$ codewords of length two. However, expression (2.5) ensures that we do not need more than this number, so the construction is possible. In fact, all the other codeword lengths can be constructed in such manner.

For the necessary part of McMillan inequality, consider the expression

$$\left(\sum_{i=1}^{s} q^{-l_i}\right)^n = \left(q^{-l_1} + q^{-l_2} + \quad \cdots \quad + q^{-l_s}\right)^n \tag{2.7}$$

where n is some positive integer. Expanding the RHS of equation (2.7) we obtain s^n terms each of the form q^{-k}, where k is a sum of codeword lengths and can take values from nl_1 to nl_s. Hence

$$\left(\sum_{i=1}^{s} q^{-l_i}\right)^n = \sum_{k=nl_1}^{nl_s} N_k q^{-k} \tag{2.8}$$

where N_k is the number of terms in the expansion of the form q^{-k}. But N_k is also the number of codewords sequences containing exactly k bits. Hence, for the code to be uniquely decodable, this number must be less than q^k, i.e.

$$\left(\sum_{i=1}^{s} q^{-l_i}\right)^n \leq \sum_{k=nl_1}^{nl_s} q^k q^{-k} \leq nl_s \tag{2.9}$$

Since the inequality given by expression (2.9) must hold for all n, including very large n, then

$$\sum_{i=1}^{s} q^{-l_i} \leq 1 \tag{2.2}$$

■

2.3.3. Instantaneously Decodable Variable-Length Codes

For practical applications, it is required that the decoding delay be as small as possible. The minimum decoding delay possible is given when a codeword is decodable as soon as it is completely received. Anything less than this would imply that the code is redundant. A code with such a property is called an *instantaneously decodable code*. It is obvious that for a code to have this property, a codeword cannot be a prefix of another codeword. Hence, these codes are also known as *prefix codes*. Hence, prefix codes are uniquely decodable with decoding delay at most l_s (maximum codeword length).

Interestingly enough, McMillan's inequality given by (2.2) is also a necessary and sufficient condition for the existence of a prefix code with codeword lengths $l_1, l_2, \cdots, l_s$. In this case, it is better known as the *Kraft inequality* [Kraft, 1949]. Chronologically, the proof of this inequality came before that of McMillan's.

2.3.4. Exhaustive Codes

A code C is said to be exhaustive iff any $\mathbf{x} \in X^+$ can be unambiguously decomposed into a sequence of codewords ending with a complete codeword or a prefix of a codeword, i.e. $\mathbf{x} = \mathbf{c}_{i_1}\mathbf{c}_{i_2}\cdots\mathbf{c}_{i_m}\mathbf{w}$ for some i, where $\mathbf{c}_{i_j} \in C$, $j = 1, 2, \cdots, m$ and $\mathbf{wy} \in C$ for some $\mathbf{y} \in X^*$. Note that an exhaustive code is uniquely decodable iff it is also a prefix code.

2.3.5. Code Efficiency and Redundancy

Shannon's first theorem [Shannon, 1948] states that the average information of a source symbol is $H(A)$, the source entropy, given by

$$H(A) = -\sum_{i=1}^{s} P(a_i)\log_q P(a_i)\,, \tag{2.10}$$

and that, for a uniquely decodable code, $L_{\text{average}} \geq H(A)$.

Accordingly, the *code efficiency*, η, is defined as

$$\eta = \frac{H(A)}{L_{\text{average}}}\,, \tag{2.11}$$

while the *code redundancy* is defined as

$$\text{Redundancy} = 1 - \eta = \frac{L_{\text{average}} - H(A)}{L_{\text{average}}}. \tag{2.12}$$

Given a memoryless source A, Huffman [1952] derived an algorithm to construct a code with the maximum possible efficiency. Codes constructed using this algorithm are known as Huffman codes, which besides having the minimum possible redundancy, are also exhaustive codes (and hence have the prefix property [Stiffler, 1971]).

2.4. Synchronisation

Synchronisation is of fundamental importance in digital communications. There are basically three levels of synchronisation that need to be taken care of. At the most basic level the receiver must have *phase* (in the case of coherent detection) or *frequency* (in the case of non-coherent detection) *synchronisation* with the carrier wave. The next level of synchronisation required is that of *symbol synchronisation*. This is required at the receiver so that the symbol detection interval is accurately aligned to that in the carrier, otherwise

the ability to make accurate symbol decisions will be degraded. In most communication systems, an even higher level of synchronisation is required, termed *frame synchronisation.* Loss of frame synchronisation is said to occur when the decoder does not correctly determine codeword boundaries. Here, we are only interested in the latter type of synchronisation and from now onwards the term synchronisation will be understood to mean *frame* or *codeword synchronisation.*

We may consider two types of synchronisation problems.

1. At the start of the transmission, the receiver loses the initial channel symbols, and hence the decoder does not know where is the start of the first complete codeword received.
2. The decoder is assumed to be already in synchronisation. However, noise on the channel causes errors in the symbols supplied to the decoder, possibly resulting in loss of synchronisation. There are three types of errors which need to be considered.
   (i) Substitution error (e.g. a '0' transformed to a '1' and vice-versa).
   (ii) Deletion error (a code symbol in the original message is deleted).
   (iii) Insertion error (an extra code symbol is inserted in the decoded message).

We can consider case (1) above as initial acquisition of synchronisation. This may be treated separately from case (2) since other mechanisms may be brought into play to acquire synchronisation, depending on the transmission protocol being used. For example, in HDLC (High-level Data Link Control) [Tanenbaum, 1988] a special flag sequence (01111110) is transmitted continuously before the start of the actual message to facilitate synchronisation.

On the other hand, acquisition of synchronisation may be considered as a special case of (2). In this case, the initial code symbols in the received message may be considered to have been deleted.

On channels without feedback, for instance, the decoder must acquire synchronisation, and maintain it, without notifying the transmitter of loss of synchronisation. In this case, it is required that the system can automatically regain synchronisation. Here, we are only going to consider the latter type of scenario. There are

several schemes one may adopt to achieve this objective, depending if fixed or variable-length codes are being used.

Case (1) applies both for fixed and variable-length codes. However, case (2) is not equally applicable for both types of codes. If the decoder is in synchronisation and the noise on the channel causes a symbol to be corrupted into another symbol (substitution error), then in the case of fixed-length codes, no loss of synchronisation occurs. However, this is not so in the case of variable-length codes. A substitution error may cause a codeword of length l_i to be decoded as a codeword of length l_j, with $l_i \neq l_j$. This will cause a loss of synchronisation. Deletion and insertion errors may cause loss of synchronisation in both fixed and variable-length codes.

2.4.1. Synchronisation Schemes using a Marker

One of the simplest schemes to adopt in order to acquire and maintain synchronisation, is to periodically insert a special symbol $\notin X$, called a *sync pulse* or *marker*. Each time the decoder receives this special marker, then this will indicate that the next symbol is the start of a codeword. Hence, if the decoder is out of synchronisation, it will acquire synchronisation as soon as it receives a marker. To improve the performance, it may also be necessary that the energy content for the sync pulse be higher than that for the other symbols, to ensure a high probability of detection. The disadvantage of this simple system is that of channel efficiency[1]. For instance, if the code is binary, the inclusion of a third symbol for the marker will, at best, give an efficiency of 63.1%, even for long frames (i.e. infrequent transmission of the marker) [Scholtz, 1980].

This scheme is slightly more involved with variable-length codes, since in this case the insertion of the sync pulse cannot be periodic. Bedi *et al.* [1992] suggest inserting a synchronisation pulse every n codewords. The decoder then will synchronise every n codewords with the help of the synchronisation pulses. Between two synchronisation pulses, however, the decoder still may lose synchronisation. The authors suggest using a

[1] Channel efficiency is defined as the $\dfrac{\text{Transmission Rate}}{\text{Channel Capacity}}$.

decoder which chooses the best n-codeword sequence among all possible such sequences. However, this could become quite impractical for large n. This technique still suffers from the same efficiency problem.

An extension to this idea is to replace the single special symbol marker with a sequence of m symbols $\in X$. Using this m-symbol sequence (also referred to as a *comma*) as a marker will now improve the channel efficiency while increasing the complexity of the decoder. Again this marker ought to be repeated periodically within a message. Several schemes may be employed in this case. For instance, it may be enforced that the marker will not appear within the data, using what Stiffler [1971] calls *comma codes*. However, in the case of channels with errors, where we may have the situation that a particular error pattern causes a codeword to be transformed into a comma, this requirement may be relaxed without much loss in performance. In this case, several occurrences of the comma must be observed to maintain synchronisation. This ideally entails the use of fixed-length codes in order to maintain the required periodicity.

2.4.2. Synchronisable Codes

If the insertion frequency of the comma is such that it occurs every codeword, then we have what are called *prefixed comma-free codes* [Ramamoorthy & Tufts, 1967]. This idea was first introduced by Gilbert [1960]. Here, a special prefix $\mathbf{p}$ of length l_p is used to mark the codeword boundaries. Each codeword in a prefixed comma-free code C is of the form $\mathbf{pw}$ where $\mathbf{w}$ is a word over X^+ of fixed-length l_w. The code is constructed such that $\mathbf{p}$ will be some distance d (≥ 1) from all l_p-bit sub-sequences in C^+. In this case the code C is said to be synchronisable.

Definition 2.1: A code C is said to be synchronisable with finite delay Z iff it is uniquely decodable and if there exists an integer Z such that if $\mathbf{x} \in X^+$, $|\mathbf{x}| \geq Z$, and $\mathbf{yxz} \in C^+$, then $\mathbf{x}$ has at least one decomposition $\mathbf{x} = \mathbf{x}_1\mathbf{x}_2$ such that either $\mathbf{yx}_1 \in C^+$ and $\mathbf{x}_2\mathbf{z} \in C^*$ or $\mathbf{yx}_1 \in C^*$ and $\mathbf{x}_2\mathbf{z} \in C^+$; in words, iff the decoder can determine a codeword boundary in a sequence of codewords consisting of at least Z code symbols, given that the start of the sequence is a suffix of a codeword in C.

Necessary and sufficient conditions for synchronisability and an algorithm to determine the synchronisation delay are given by Capocelli [1979], who gives a simplified unified treatment for unique decodability, decoding delay and synchronisability of codes.

For a synchronisable code, as defined in Definition 2.1, if one or more code symbols are lost (but not a complete codeword), the following two outcomes may be possible.

(1). The received sequence is not decodable.

(2). The received sequence is decodable, but with the property that correct synchronisation is achieved automatically after the first few words are decoded.

For codes which exhibit property (1) above, we have the advantage that we have an indication of loss of synchronisation. In this case, we may simply request a re-transmission, if there is a feedback channel. Otherwise, we need to discard symbols from the received sequence until the sequence is decodable again, at which point it is hoped that synchronisation has been regained. Note that in this case the code cannot be exhaustive.

For codes which exhibit property (2), we have the advantage that no extra control logic is required at the decoder to acquire synchronisation. Consequently, such codes are called self-synchronising. However, this is achieved at the expense of losing the ability to detect the out of synchronisation condition.

Note that any given code may exhibit both properties for different sequences. However, self-synchronising codes cannot be of fixed-length, since in this case, if the decoder is out of synchronisation, unless there is another error it will continue to misplace the codeword boundaries indefinitely.

The prefixed comma-free codes mentioned above are an example of codes with property (1). In this case, it is easy to see that the synchronisation delay is l_p+l_w [Scholtz, 1980].

2.4.2.1. Comma-Free Codes

Another example of codes which exhibit property (1) are the *comma-free* codes introduced by Golomb *et al.* [1958], of fixed length L.

Definition 2.2: A fixed length code C is said to be *comma-free* if for every pair of words $\mathbf{c}_i = c_{i_1}c_{i_2}\cdots c_{i_L}$ and $\mathbf{c}_j = c_{j_1}c_{j_2}\cdots c_{j_L}$, $\mathbf{c}_i$, $\mathbf{c}_j \in C$, the words $c_{i_k}c_{i_{k+1}}\cdots c_{i_L}c_{j_1}c_{j_2}\cdots c_{j_{k-1}}$, $k = 2, 3, \cdots, L$, are not in C.

Hence, following this definition, it is easy to see that when the decoder is out of synchronisation, the received sequence will not be decodable. To regain synchronisation the decoder simple deletes symbols consecutively until it can decode a sequence of codewords. In this case the synchronisation delay is $2L-2$. Notice that in the original definition the code is assumed to be of fixed-length. Scholtz [1969] gives a simple construction algorithm for comma-free codes with a maximum number of codewords. In this paper, he also extends the notion of comma-free codes to variable-length codes and gives a construction procedure for these codes. Again, in the case of variable-length comma-free codes, the decoder must scan the received sequence to find the correct synchronisation.

Kendall and Reed [1962] discuss a sub-class of fixed-length comma-free codes called *path invariant comma-free codes* which turn out to be relatively easy to encode and decode, and with synchronisation delay of L. However, these codes are not very efficient.

Levy [1966] extends the idea of comma-free codes to error-correcting codes. If a suitable modification vector[2] is added to a cyclic error-correcting code (with sufficient redundancy), a comma-free code may be generated. Hence, a simple method can be devised whereby the relative decoding simplicity of cyclic error-correcting codes is retained, while their synchronisability capability is greatly enhanced, with minimal overheads (just the addition of a vector at the transmitter and at the receiver) (see also Tavares and Fukada [1969]). For less redundant codes Levy considered a relaxation of the comma-freedom required by specifying code parameters $[z, \chi]$. Here, z is the maximum synchronisation slip considered, and χ is the minimum Hamming distance of the resultant

[2] A modification vector is a word of the same length as a codeword but which itself is not a codeword. When added to all codewords in a code, it modifies the code while retaining the same distance properties of the original code.

(out of synchronisation) words, to the codewords in the code. If z is specified equal to the codeword length, then χ gives the *index of comma freedom*.

2.4.3. Statistically Synchronisable Codes

The codes studied in the previous section are also known as *bounded synchronisation delay* (BSD) *codes*, since the number of code symbols, z, that need to be observed to achieve synchronisation is bounded by Z. Another class of codes which has been studied extensively is that of *statistically synchronisable codes* [Gilbert & Moore, 1959] [Stiffler, 1971] [Wei & Scholtz, 1980] [Capocelli *et al.*, 1988]. Here, the synchronisation delay is not bounded. However, with probability one, the code will synchronise if a large enough number of code symbols is observed; i.e. for statistically synchronisable codes, the probability of synchronising after receiving z code symbols is

$$\lim_{Z\to\infty} \Pr\{z \leq Z\} = 1. \tag{2.13}$$

Here it is assumed that the source can generate all possible messages (an *$\in$-guaranteed message source* [Wei & Scholtz, 1980]).

Capocelli *et al.* [1988] have shown that a code is statistically synchronisable iff it has at least one synchronising sequence. This is a sequence of codewords that is not a sub-string of any other sequence of codewords. Hence, whenever this synchronising sequence of codewords appears within the message, the decoder will always re-synchronise (hence the necessity of having an $\in$-guaranteed source). Capocelli *et al.* [1988] give an algorithm to test whether a code is statistically synchronisable and this is simplified for the special case of prefix codes. Neumann [1962a, 1962b, 1964] considers the construction of such codes having what he calls a *synchronising input sequence*. Hatcher [1969] gives error-correcting capability to these codes.

When the synchronising sequence consists of just a single codeword, the code is called *synchronous* [Ferguson & Rabinowitz, 1984]. In this case, all that is required for the code to be statistically synchronisable is that the synchronising codeword occurs with non-zero probability.

A codeword $\mathbf{c}_i = c_{i_1}c_{i_2}\cdots c_{i_{l_i}}$ is synchronising for code C if it satisfies the following two conditions.

1. For all codewords $\mathbf{c}_j = c_{j_1}c_{j_2}\cdots c_{j_{l_j}}$ in C with $l_j > l_i$, in which $\mathbf{c}_i$ appears as a sub-sequence, then $\mathbf{c}_i$ must be the suffix part of $\mathbf{c}_j$.
2. If a prefix of $\mathbf{c}_i$ can form part of a suffix of another codeword, then the remaining part of $\mathbf{c}_i$ must form a sequence of codewords.

In this case, when $\mathbf{c}_i$ is received without any errors, the decoder will always re-synchronise and start decoding correctly from the first codeword following $\mathbf{c}_i$.

Since in the case of statistically synchronisable codes, the synchronising delay is not bounded, then it is of interest to know the average number of code symbols which need to be observed before synchronisation is achieved. We shall call this the *average synchronisation delay* of the code. Another possible measure for the synchronisation capability of the code is the *average error span* [Maxted & Robinson, 1985], which is the average number of source symbols lost during the re-synchronisation process.

Ferguson and Rabinowitz [1984] argue that to reduce the error-propagation (i.e. to reduce the average synchronisation delay or the average error-span) in a code, the code must be designed such that the probability of receiving a synchronising codeword is maximised. This can be achieved by devising a code with more synchronising codewords and/or shorter (more probable) ones. Constructing codes with such a property still remains an open problem.

Note that the presence of synchronising codewords may not necessarily decrease the code efficiency. In fact, synchronous Huffman codes may be designed using a heuristic algorithm [Ferguson & Rabinowitz, 1984]. In this case, the synchronous code will also be optimal. Deterministic construction algorithms for synchronous variable-length codes, which are, however, sub-optimal (with average length slightly larger than that obtained by the Huffman algorithm) are given by Montgomery and Abrahams [1986] and Capocelli *et al.* [1992]. In many cases, the resulting codes have better statistical synchronising performance than the corresponding optimal synchronous codes.

Montgomery and Abrahams [1986] have pointed out a further complication to the determination of the code with the "best" synchronisation properties. Re-synchronisation is attained not only through the occurrence of a synchronising codeword, but also if the sequence corresponding to a synchronising codeword occurs somewhere else within the message. For example, for the code given in Table 2.1, the codewords 010 and 0110 are both synchronising. However, the sequence 010, say, may also occur when *ab* (0010) is transmitted. Hence, whereas the probability of the two synchronising codewords is 0.190, the probability that a synchronising sequence occurs is 0.325. In addition, there may be other synchronising sequences which are not codewords.

Source Symbol	Probability	C_1
a	0.28	00
b	0.26	10
c	0.13	010
d	0.12	110
e	0.06	0110
f	0.05	0111
g	0.05	1110
h	0.05	1111

Table 2.1: Synchronous code for eight-symbol source

Accordingly, Titchener [1988] has questioned the validity of this model for the synchronisation recovery of a variable-length code and has proposed an iterative construction algorithm for statistically synchronisable codes. The resultant codes are called *T-codes*. The construction is based on the idea that if the starting code is statistically synchronising, then by deleting a codeword from this original code and then using this codeword to build new codewords, the resultant code will also be statistically synchronisable [Titchener, 1984]. The construction is similar to that given by Scholtz [1966] for BSD codes. However, Titchener's construction ensures an exhaustive code.

Maxted and Robinson [1985] (see also [Monaco & Lawler, 1987]) use a state model to derive an expression for the error span of variable-length codes for single bit errors. Rahman and Misbahuddin [1989] extend this model to give the performance of variable-

length codes on the BSC. This is further enhanced by Takishima *et al.* [1994] who give a numerical method to quickly evaluate the error span, both for a single bit error and for the BSC. They also observed, as pointed to above, that maximising the probability of receiving a synchronising codeword does not always produce the code with the minimum error span. Consequently, they give a heuristic algorithm for constructing good self-synchronising codes better than the ones given by Ferguson and Rabinowitz [1984] and Montgomery and Abrahams [1986].

2.5. α-Correcting Codes

In the previous sections we have highlighted some properties of variable-length codes. However, with the exception of Section 2.4, the noiseless case was always assumed. Even in Section 2.4, we have basically assumed that there is a single error event on the channel which causes loss of synchronisation, however in the re-synchronisation process it is assumed that no further errors occur.

The first work found in the literature concerning variable-length codes which are also designed to combat channel noise is that done by the Coding Group at Parke Mathematical Laboratories from 1957 to 1968. Most of this work is reported by Hartnett [1974]. We shall call such codes, in general, *variable-length error-correcting* (VLEC) codes.

Here, we shall be dealing mainly with their work concerning channel models in which only substitution errors are allowed. This model is one which allows a set of permissible error-patterns based on the code being used. This turns out to be a big disadvantage when dealing with variable-length codes, as will be shown in Chapter 3. The problem here is that they treat VLEC codes as block codes whereas, as will be shown, they are really trellis codes, exhibiting a form of "memory".

In a *t-error-correcting* block code, all *t*-bit (or less) error patterns are correctable. If the code is not perfect, then we may allow other arbitrary error patterns (with the minimum possible weight) to be also correctable [MacWilliams & Sloane, 1978]. For instance, the code {00000, 01110, 10011, 11101} has Hamming distance three [Sweeney, 1991]. This implies that it can correct all single bit error-patterns. However, since this is not a perfect

code, it can also correct some double-bit error patterns. From the possible remaining error-patterns, one may arbitrarily choose the set {01001, 00101} with minimum weight. Hence, the correctable error patterns for this code are {00000, 00001, 00010, 00100, 01000, 10000, 01001, 00101}. This set of error-patterns spans the whole 5-tuple space, and hence any 5-tuple may be decoded into one of the four codewords. Hence, as an example, if any of the words {01110, 0111**1**, 011**0**0, 01**0**10, 0**0**110, **1**1110, 0**0**11**1**, 01**0**1**1**}[3] are received, the decoded codeword will be 01110. The reason why we choose these as our correctable error-patterns is that, over the BSC, these are the most probable ones, and hence by choosing these we will be minimising the probability of making a decoding error and thus of achieving maximum likelihood decoding.

Calabi and Arquette [1974a] have adapted this notion to variable-length codes. They define an *admissibility* (or *error*) *mapping* α on the codewords of a VLEC code C, such that if a received word $\mathbf{w} \in \alpha(\mathbf{c}_i) \subset X^+$, then $\mathbf{w}$ will be decoded as $\mathbf{c}_i$, where $\mathbf{c}_i \in C$. They also define the *expanded code* of C under the admissibility mapping α as $E = \bigcup_{i=1}^{s} \{\alpha(\mathbf{c}_i) : \mathbf{c}_i \in C\}$. Then, in order for C to be decodable in noise (under the α-admissibility mapping), E must be uniquely decodable [Calabi & Arquette, 1974b]. This is only true if α allows only substitution errors, in which case all the words in $\alpha(\mathbf{c}_i)$ will be of the same length as $\mathbf{c}_i$.

Sato [1979] and Capocelli [1982] have considered something similar, which they called *multi-level encodings*. In particular, they have shown that codeword decomposition is not equivalent to decodability. To illustrate this, consider that we have a two-symbol source $\{a, b\}$ and that the code alphabet is ternary, i.e. $X = \{0, 1, 2\}$. If the two words {0012, 00} are mapped to a and {012, 12012} are mapped to b, then the expanded code {0012, 00, 012, 12012} is not uniquely decodable. However, we can still perform "error-correction" under this mapping. As an example, if the message 0012012 is received, this could be decomposed either as 0012.012 or as 00.12012. However, both these decompositions will result in the decoded message ab. Notice here that the mapping is

[3] For clarity, the corrupted bits are shown in bold.

allowing insertion/deletion errors as well. In this case it will be even more difficult to characterise the admissibility mapping required.

Again assuming that only substitution errors are allowed, the problem still remains of how best to define the admissibility mapping. This was relatively easy to do in the case of fixed-length codes, because for maximum likelihood decoding we must choose α such that those error-patterns with minimum weight are selected, since these are the most likely on the BSC. However, this choice of α becomes problematic in the case of variable-length codes, even for the simple case of the BSC. Calabi and Arquette [1974a] opted for an arbitrary choice, whereby α is a function of the codeword, with α_t denoting that the mapping can correct all t-bit (or less) error-patterns[4]. In general, t will be a function of the codeword chosen.

Example 2.1: Consider the α-correcting code C_2 given in Table 2.2. Then $\alpha_1(a) = \{000, 001, 010, 100\}$, i.e. $\alpha_1(a)$ contains the codeword for a plus all words which vary by one bit from this codeword. Similarly $\alpha_2(b)$ will contain the codeword for b, and all words which vary from this codeword by at most two bits. Thus there are 4 words in $\alpha_1(a)$ and 37 words in $\alpha_2(b)$. Applying the Sardinas and Patterson [1953] test on the expanded code $\alpha_1(a) \cup \alpha_2(b)$ shows that this is uniquely decodable, and hence C_2 is α-correcting and will correct all single bit errors in a and all single and double bit errors in b.

Source Symbol	Codeword
a	000
b	00011111

Table 2.2: An α-correcting code C_2

There are two main problems with this notion. The expanded code may not always be exhaustive, as in the case of Example 2.1. The corresponding property in the case of fixed-length codes is that not all codes are perfect. However, in the case of fixed-length codes it is relatively easy to either output an erasure in the case of a detectable error-pattern

[4] In fact they also discuss other mappings suitable for different kinds of channels.

of more than t-bits, or, as indicated earlier, correct some error-patterns with more than t-bits. In the case of variable-length α-correcting codes, should an error pattern exceed the correction capability of the code, it is difficult to determine which codeword length to decode. One possibility is to output erasures until a word in the expanded code is detected. However, the error-propagation caused by this loss of synchronisation will have a negative effect on the overall performance of the code. The second problem is that the decoding delay may be unbounded, as is the case for the code given in Example 2.1, for instance. These problems may be partly solved, by enforcing that the code will be instantaneously decodable.

2.6. α-Prompt Codes

A code C is *α-prompt* iff for any two distinct codewords $\mathbf{c}_i$, $\mathbf{c}_j$, no sequence in $\alpha(\mathbf{c}_i)$ is a prefix of a sequence in $\alpha(\mathbf{c}_j)$; in other words, iff the expanded code E of C is a prefix code. Hence, α-prompt codes are instantaneously decodable (under the mapping α) VLEC codes. An example of an α_1-prompt code for the 26-symbol English alphabet is given in Table A.1 which is reproduced, with slight modification, from Calabi and Arquette [1974a].

Consider the variable-length code C ($s_1@L_1$, $s_2@L_2$, $\cdots$, $s_\sigma@L_\sigma$). Let $\mathbf{c}_i = c_{i_1}c_{i_2}\cdots c_{i_{L_j}}$, where $\mathbf{c}_i \in C$ and $|\mathbf{c}_i| = L_j$. Then, we define the *prefix decomposition* of $\mathbf{c}_i$ to be the set $\{\mathbf{p}_{i_1}, \mathbf{p}_{i_2}, \cdots, \mathbf{p}_{i_j}\}$, where $\mathbf{p}_{i_k} = c_{i_1}c_{i_2}\cdots c_{i_{L_k}}$, for all $k = 1, 2, \cdots, j$. In particular, note that $\mathbf{p}_{i_j} = \mathbf{c}_i$. We also define the prefix decomposition of C to consist of the sets $\{P_1, P_2, \cdots, P_\sigma\}$, where $P_j = \bigcup_{i=\bar{s}_j+1}^{s} \mathbf{p}_{i_j}$.

2.6.1. Prefix Decoding Algorithm

Since in the case of α-prompt codes, the related expanded code is instantaneously decodable, then it is much easier to formulate a decoding algorithm for α-prompt codes, even including error patterns which are not in the admissibility range α, than it was for the case of α-correcting codes. A possible complete decoder for an α-prompt code C ($s_1@L_1$,

$s_2@L_2, \cdots, s_\sigma@L_\sigma)$ with prefix decomposition $\{P_1, P_2, \cdots, P_\sigma\}$ over the BSC is the following.

1. Take the next L_1 bits from the received bit sequence and decode these to a prefix from the set P_1 using the α-admissible mapping. If no word from P_1 is found satisfying this mapping, then choose that prefix at the minimum Hamming distance to the received sequence having the minimum index[5]. Let the decoded prefix be $\mathbf{p}_{i_1}$.
2. If $\mathbf{p}_{i_1} \in C$, then decode the source symbol corresponding to $\mathbf{c}_i = \mathbf{p}_{i_1}$ and go to Step 1. Otherwise, let $j = 2$ and go to Step 3.
3. Take the previous L_{j-1} bits considered in the previous steps and the next $L_j - L_{j-1}$ bits from the received bit sequence (i.e. L_j bits in all) and decode these to a prefix from the set P_j using the α-admissibility mapping. Again, if no word from P_j is found satisfying this mapping, then choose that prefix at the minimum Hamming distance to the received sequence having the minimum index. Let the decoded prefix be $\mathbf{p}_{i_j}$.
4. If $\mathbf{p}_{i_j} \in C$, then decode the source symbol corresponding to $\mathbf{c}_i = \mathbf{p}_{i_j}$ and go to Step 1. Otherwise, increment j and go back to Step 3.

We shall call the above algorithm *prefix decoding of α-prompt codes*. Note that the above algorithm always decodes to a codeword, even if the error pattern is not α-admissible.

2.6.2. Segment Decomposition

Consider the variable-length code C $(s_1@L_1, s_2@L_2, \cdots, s_\sigma@L_\sigma)$. Let $\mathbf{c}_i = c_{i_1}c_{i_2}\cdots c_{i_{L_j}}$, where $\mathbf{c}_i \in C$ and $|\mathbf{c}_i| = L_j$. Then, the *segment decomposition* of $\mathbf{c}_i$ is defined to be the set $\{\mathbf{q}_{i_1}, \mathbf{q}_{i_2}, \cdots, \mathbf{q}_{i_j}\}$ where, $\mathbf{q}_{i_1} = c_{i_1}c_{i_2}\cdots c_{i_{L_1}}$, $\mathbf{q}_{i_2} = c_{i_{L_1+1}}\, c_{i_{L_1+2}}\cdots c_{i_{L_2}}$, $\cdots$, $\mathbf{q}_{i_j} = c_{i_{L_{j-1}+1}}\, c_{i_{L_{j-1}+2}}\cdots c_{i_{L_j}}$ [Bernard & Sharma, 1988][6]. Hence, $\mathbf{c}_i = \mathbf{q}_{i_1}\mathbf{q}_{i_2}\cdots\mathbf{q}_{i_j}$. We shall define the segment decomposition of C to be the sets $\{Q_1, Q_2, \cdots, Q_\sigma\}$ where $Q_j = \bigcup_{i=\bar{s}_j+1}^{s} \mathbf{q}_{i_j}$. Bernard and

---

[5] The minimum Hamming distance requirement will ensure that the most likely prefix is chosen, whereas the requirement that the minimum index prefix is chosen will ensure that the shortest (most probable) prefix is chosen.

[6] The definition given by Bernard and Sharma is essentially the same as the one given here. However, in their case they use l_i instead of L_j for the length of codeword $\mathbf{c}_i$. Since in general there will be more than one codeword with the same length in the code, then their definition effectively allows for empty segments, whereas ours does not.

Sharma [1988] define $\alpha_{t_1,t_2,\cdots,t_\sigma}$-prompt codes that can correct t_1 substitution errors in the first segment, t_2 substitution errors in the second segment and so on. This admissibility mapping is different from the one considered earlier, where the number of correctable errors was defined per codeword, whereas here they are defined per segment. Consequently the decoding algorithm will be slightly different and will be given in Section 2.6.3.

Some combinatorial results are possible using this mapping, which unifies some theories in Coding Theory from the areas of noiseless coding and error-correction coding.

Theorem 2.2: An $\alpha_{t_1,t_2,\cdots,t_\sigma}$-prompt code C ($s_1@L_1$, $s_2@L_2$, $\cdots$, $s_\sigma@L_\sigma$) must satisfy the condition

$$\sum_{i=1}^{\sigma} s_i |r_\alpha|_{L_i} q^{-L_i} \leq 1 \tag{2.14}$$

where $|r_\alpha|_{L_i}$ is the effective range of a codeword, $\mathbf{c}_j$, of length L_i; i.e. the number of words in $\alpha(\mathbf{c}_j)$, given by

$$|r_\alpha|_{L_i} = \left|\alpha_{t_1}\right|_{L_1} \left|\alpha_{t_2}\right|_{L_2-L_1} \cdots \left|\alpha_{t_i}\right|_{L_i-L_{i-1}} \tag{2.15}$$

and

$$\left|\alpha_{t_i}\right|_{L_i} = \sum_{k=0}^{t_i} \binom{L_i}{k} (q-1)^k \tag{2.16}$$

Theorem 2.3: An $\alpha_{t_1,t_2,\cdots,t_\sigma}$-prompt code C ($s_1@L_1$, $s_2@L_2$, $\cdots$, $s_\sigma@L_\sigma$) exists if

$$\sum_{i=1}^{\sigma} s_i \left|\alpha_{2t_i}\right|_{L_i} q^{-L_i} \geq 1 \tag{2.17}$$

Both Theorems 2.2 and 2.3 are proved by Bernard and Sharma [1988]. Theorem 2.2 reduces to the Hamming sphere packing upper bound in the case of fixed-length error-correcting codes, and to the necessary part of Kraft's inequality for the noiseless case. Whereas, Theorem 2.3 reduces to the Gilbert lower bound for the case of fixed-length error-correcting codes, and to the sufficiency part of Kraft's inequality for the noiseless case. Further, Bernard and Sharma [1990] prove the following theorem.

Theorem 2.4: The average length, L_{average}, of an $\alpha_{t_1,t_2,\cdots,t_\sigma}$-prompt code C ($s_1@L_1$, $s_2@L_2$, $\cdots$, $s_\sigma@L_\sigma$) when used to encode a source A with symbol probabilities $P(a_1)$, $P(a_2)$,

$\cdots$, $P(a_s)$ is bounded by

$$L_{\text{average}} \geq \sum_{i=1}^{s} P(a_i) \log_q \left(\frac{|r_\alpha|_{l_i}}{P(a_i)} \right) \tag{2.18}$$

with equality iff

$$l_i = \log_q \left(\frac{|r_\alpha|_{l_i}}{P(a_i)} \right) \text{ for all } i \tag{2.19}$$

where l_i is the length of codeword $\mathbf{c}_i$ and s is the total number of codewords.

Theorem 2.4 reduces to Shannon's first theorem for the noiseless case.

Bernard and Sharma [1992] use this per segment error-mapping to define perfect $\alpha_{t_1,t_2,\dots,t_\sigma}$-prompt codes.

Definition 2.3: An $\alpha_{t_1,t_2,\dots,t_\sigma}$-prompt code satisfying (2.14) with equality can correct up to t_1 random errors in the first segment and no more, up to t_2 random errors in the second segment and no more, and so on for each segment. Such a code is called a perfect $\alpha_{t_1,t_2,\dots,t_\sigma}$-prompt code.

In their paper, they give several examples of perfect $\alpha_{t_1,t_2,\dots,t_\sigma}$-prompt codes derived from fixed-length perfect codes. Their construction is very simple. Suppose that C_1 and C_2 are two perfect fixed-length error-correcting codes, capable of correcting up to t_1 and t_2 substitution errors, of length L_1 and L_2, and with S_1 and S_2 codewords, respectively. Note that C_1 and C_2 need not necessarily be different codes. Then, a perfect α_{t_1,t_2}-prompt code C_3 may be constructed by taking s_1 codewords from C_1 to form length L_1 codewords in C_3. Each one of the remaining codewords in C_1 (S_1-s_1 in all) is then used as prefix to all the codewords in C_2 to form $(S_1-s_1)\times S_2$ codewords of length L_1+L_2. Thus, C_3 will be a (s_1@L_1, $[S_1-s_1]\times S_2$@L_1+L_2) perfect α_{t_1,t_2}-prompt code. An example of a perfect $\alpha_{1,1}$-prompt code for the 26-symbol English source is shown in Table A.1. This was constructed using the (7,4) Hamming code for C_1 and the (3,1) repetition code for C_2, both perfect fixed-length codes with minimum distance 3. This construction can easily be extended to produce VLEC codes with multiple lengths.

2.6.3. Segment Decoding Algorithm

Since the admissibility mapping is different for the $\alpha_{t_1,t_2,\cdots,t_\sigma}$-prompt codes defined by Bernard and Sharma [1988], then the decoding algorithm given in Section 2.6.1 for α-prompt codes must be altered.

Assume a variable-length $\alpha_{t_1,t_2,\cdots,t_\sigma}$-prompt code C ($s_1@L_1$, $s_2@L_2$, $\cdots$, $s_\sigma@L_\sigma$) with segment decomposition $\{Q_1, Q_2, \cdots, Q_\sigma\}$.

1. Let $j = 1$.
2. Take the next L_j bits from the received bit sequence and decode these to a segment from the set Q_j using the α_{t_j}-admissibility mapping. If no word from Q_j is found satisfying this mapping, then choose that segment at the minimum Hamming distance to the received sequence having the minimum index. Let the decoded segment be $\mathbf{q}_{i_j}$.
3. If $\mathbf{q}_{i_1}\mathbf{q}_{i_2}\cdots\mathbf{q}_{i_j} \in C$, then decode the source symbol corresponding to $\mathbf{c}_i = \mathbf{q}_{i_1}\mathbf{q}_{i_2}\cdots\mathbf{q}_{i_j}$ and go to Step 2. Otherwise, increment j (until $j \le \sigma$, see below) and go back to Step 2.

We shall call the above algorithm *segment decoding of $\alpha_{t_1,t_2,\cdots,t_\sigma}$-prompt codes*.

One problem that may occur in the above algorithm is that a decoded sequence of segments $\mathbf{q}_{i_1}\mathbf{q}_{i_2}\cdots\mathbf{q}_{i_j}$ may not form a valid codeword prefix, since in this case the segments are being decoded independently. Hence, j will reach σ before any codeword is decoded, at which point there are no more segments to decode. One way round this problem is as follows. Suppose that $\mathbf{q}_{i_1}\mathbf{q}_{i_2}\cdots\mathbf{q}_{i_{j-1}}$ is a valid prefix to a codeword, while $\mathbf{q}_{i_1}\mathbf{q}_{i_2}\cdots\mathbf{q}_{i_j}$ is not[7]. Then, instead of decoding $\mathbf{q}_{i_j}$ following the above algorithm, this is decoded by choosing the minimum distance segment from the list of segments arising only from those codewords whose prefix is $\mathbf{q}_{i_1}\mathbf{q}_{i_2}\cdots\mathbf{q}_{i_{j-1}}$. Another possibility is to emit an erasure symbol, if incomplete decoding is allowed. However, even here, there is the problem of determining the number of segments in the erasure symbol, since this will affect the decoding of the subsequent codewords.

[7] This is always possible at least for $j = 2$, since q_{i_1} is always a codeword prefix or a codeword.

2.7. Two-Length Error-Correcting Codes

Using similar concepts, Dunscombe [1988] defines two-length t-prefix codes capable of correcting t bit errors in each codeword. Dunscombe proposes that the length of the codewords should be a multiple of each other and conjectures that the best performance is achieved when the length of the long codewords is double that of the short codewords. These codes are really $\alpha_{t,t}$-prompt codes as defined by Bernard and Sharma [1988]. However, the decoding algorithm given by Dunscombe is essentially equivalent to the prefix decoding algorithm presented in Section 2.6.1.

Dunscombe also gives a construction algorithm for two-length t-prefix codes which is essentially the same as the one given by Bernard and Sharma [1992] to construct their perfect $\alpha_{t_1,t_2,\ldots,t_\sigma}$-prompt codes. The only difference is that the starting codes C_1 and C_2 are really the same code (called the *base code*) and this is not necessarily perfect. The base code must be a t-error-correcting code, of fixed length L. If the construction procedure outlined in Section 2.6.2 is followed exactly, then Dunscombe calls this a *complete derived code*; otherwise, if during the construction not all codewords of C_1 are used as short codewords or as prefix of long codewords, or if not all codewords of C_2 are used as suffixes to all the chosen prefixes, then these are simply called *derived codes*. Dunscombe also defines a special class of codes called *fixed-ratio derived codes* which put some restrictions on the choice of codewords (for details refer to [Dunscombe, 1988]).

Dunscombe also derives an expression for the expected number of L-bit blocks that are out of synchronisation given that a synchronisation error occurs, both for complete and fixed-ratio derived codes. This result is improved upon by Escott [1995] which gives a similar expression but in terms of the number of codewords (and hence source symbols). Escott also gives an expression for the probability of symbol error.

2.8. Instantaneous Decoding using the Massey Metric

Both decoding algorithms given in Section 2.6 are instantaneous, in the sense that as soon as a codeword is completely received, it will be decodable. This was in fact the main objective of α-prompt codes. However, the Hamming metric used in both these algorithms

is by no means optimum. The main problem with these algorithms is that notwithstanding the fact that these are supposed to be instantaneous codes, in the case of channels with errors, even if these algorithms may correct some error-patterns it does not necessarily follow that the decoding process is optimum. The main reason for this is that, should a decoding error be made on the current codeword, this will have knock on effects on subsequent decodings, due to loss of synchronisation. So obviously, once there is a synchronisation error, there will follow a relatively large number of subsequent codewords that are also erroneously decoded, even if in the meantime there were no further errors on the channel. Hence, although these codes may be decoded instantaneously, it is clear that if we were to wait for several codewords before making a decoding decision, better performance[8] should be expected. Such a decoding algorithm will be given in Chapter 3.

However, one step forward in this direction was indirectly suggested by Massey [1972]. In this paper, he derives what he calls "the required statistic for minimum-error-probability decoding of variable-length codes". As we shall see in Chapter 3, this is only true for instantaneous decoding of VLEC codes. That is, only if VLEC codes are considered to be pseudo-block codes.

In his paper, Massey converts the variable-length code C ($s_1@L_1$, $s_2@L_2$, $\cdots$, $s_\sigma@L_\sigma$) into a fixed-length code by adding a "random tail" to each codeword, such that each resultant word has length L, where $L = L_\sigma$ is the length of the maximum length codeword. Massey assumes that the random tail is selected statistically independently of the codeword and that the bits in this tail are chosen independently according to a probability measure $Q(\,)$ over the channel input alphabet. In fact, these random tails consist of bits resulting from subsequent codewords in the encoded message. Hence, the assumption that the bits in the tail are independent is not exactly true and here it will be dropped.

[8] Better in the sense that less decoding errors will be made.

Quoting from Massey's results [Massey, 1972][9], the joint probability of sending message (codeword) $\mathbf{c}_m$ and receiving $\mathbf{y}$ is given by

$$\Pr(\mathbf{c}_m,\mathbf{y}) = P_m \prod_{i=1}^{l_m} P\left(y_i \middle| c_{m_i}\right) \prod_{j=1}^{L-l_m} P_0\left(y_{l_m+j}\right) \tag{2.20}$$

where,

$$P_0(y_i) = \sum_{t_k} P(y_i|t_k) Q(t_k) . \tag{2.21}$$

Here, $\mathbf{y} = y_1y_2\cdots y_L$ is the received L bit sequence; P_m is the probability of sending codeword $\mathbf{c}_m = c_{m_1}c_{m_2}\cdots c_{m_{l_m}}$; l_m is the length of $\mathbf{c}_m$; and t_k is the bit at position k in the random tail. Hence, given $\mathbf{y}$, the optimum decoding rule using this model is to choose m' as the value of m which maximises $\Pr(\mathbf{c}_m,\mathbf{y})$. We shall call the metric given by equation (2.20) the *Massey metric*. Note that Massey goes on to derive the Fano metric from equations (2.20) and (2.21), used in sequential decoding of convolutional codes (see for example [Lin & Costello, 1983]).

Removing now Massey's assumption, let T_m be the set of possible tails for codeword $\mathbf{c}_m$ with length l_m. Then, $T_m = \{\mathbf{c}_{i_1}\mathbf{c}_{i_2}\cdots\mathbf{c}_{i_{j-1}}c_{i_{j_1}}c_{i_{j_2}}\cdots c_{i_{j_k}} : \mathbf{c}_{i_1}, \mathbf{c}_{i_2}, \cdots, \mathbf{c}_{i_{j-1}},\ c_{i_{j_1}}c_{i_{j_2}}\cdots c_{i_{j_k}}c_{i_{j_{k+1}}}\cdots c_{i_{j_l}} \in C$, and $l_{i_1} + l_{i_2} + \cdots + l_{i_{j-1}} + k = L - l_m\}$,[10] i.e. T_m is the set of all codeword combinations followed, possibly, by prefixes of codewords, with total length equal to $L - l_m$. The probability distribution of T_m will then be simply derived from the probability of the constituent codewords.

Using the above probability distribution for the "random" tail and assuming that the code is binary and that the channel is the BSC with cross-over probability p, equations (2.20) and (2.21) become

$$\Pr(\mathbf{c}_m,\mathbf{y}) = P_m p^{h_m}(1-p)^{l_m-h_m} P_{0_m} \tag{2.22}$$

where,

$$P_{0_m} = \sum_{\mathbf{t}_i \in T_m} P(\mathbf{t}_i) p^{h_{\mathbf{t}_i}} (1-p)^{L-l_m-h_{\mathbf{t}_i}} . \tag{2.23}$$

Here, h_m is the Hamming distance between the first l_m bits of $\mathbf{y}$, $y_1y_2\cdots y_{l_m}$, and $\mathbf{c}_m$; $P(\mathbf{t}_i)$ is the probability of a particular tail, $\mathbf{t}_i$, for codeword $\mathbf{c}_m$; and $h_{\mathbf{t}_i}$ is the Hamming distance between the last $L - l_m$ bits of $\mathbf{y}$, $y_{l_m+1}y_{l_m+2}\cdots y_L$, and $\mathbf{t}_i$. The metric derived from

[9] With slight change of notation.

[10] As usual, $l_{i_j} = |\mathbf{c}_{i_j}|$.

(2.22) will be called the *modified Massey metric*. The following example will clarify some of the points discussed above.

Example 2.2: Consider code C_3 given in Table 2.3. Since there are two codeword lengths in this code, then there are two "random" tail lengths. In practice there is only one, since the tail for the longer codewords is of length 0. Hence, the possible tails for both the short codewords are derived from the prefix of all codewords, of length 3. These possible tails for a and b, together with their probabilities are given in Table 2.4.

Source Symbol	Probability, $P(A_1)$	Codeword
a	0.35	00000
b	0.30	10110
c	0.10	11001111
d	0.10	01111111
e	0.05	11011010
f	0.05	01101010
g	0.03	01011001
h	0.02	11101001

Table 2.3: An eight-codeword α_1-prompt code C_3

Possible tails for a and b	Probability
000	0.35
101	0.30
011	0.15
110	0.15
010	0.03
111	0.02

Table 2.4: Possible tails for code C_3

Suppose that the bit sequence 10011110 is received, and let p = 0.01. Then, for the two short codewords, P_0 is given by

$$P_0 = \left[(0.35+0.3+0.15)\times 0.01^2 \times 0.99\right]+\left[0.15\times 0.99^3\right]+\left[(0.03+0.02)\times 0.01\times 0.99^2\right]$$

$$= 0.1461$$

Whereas for the long codewords, $P_0 = 1$.

Using equation (2.22), the calculated $\Pr(\mathbf{c}_m, \mathbf{y})$ for all possible codewords and the Hamming distance to the received bit sequence are given in Table 2.5. From this table we

can immediately deduce that the most likely symbol is e, even though the codeword for b is also at a Hamming distance of 2 and b is more likely to occur than e. However, in this case the tail 110 is not very likely and this tips the balance in favour of e.

Source Symbol	Hamming Distance	$\Pr(\mathbf{c}_m, \mathbf{y})$
a	3	5.01×10^{-8}
b	2	4.25×10^{-6}
c	3	9.51×10^{-8}
d	4	9.61×10^{-10}
e	2	4.71×10^{-6}
f	5	4.85×10^{-12}
g	5	2.91×10^{-12}
h	6	1.96×10^{-14}

Table 2.5: Probabilities for all codewords of C_3 given that we receive 10011110

Hence a third instantaneous decoding algorithm for prompt VLEC codes may now be given. This will be called the *tail decoding algorithm for α-prompt codes.*

1. Let $\mathbf{y}$ be the next L bits of the input bit sequence (where L is the maximum codeword length of the VLEC code C).
2. Calculate $\Pr(\mathbf{c}_m, \mathbf{y})$ for all codewords $\mathbf{c}_m$ in C.
3. Decode codeword $\mathbf{c}_{m'}$ such that $\Pr(\mathbf{c}_{m'}, \mathbf{y}) \geq \Pr(\mathbf{c}_m, \mathbf{y})$ for all $m=1, 2, \cdots, s$, where s is the number of codewords in C.
4. Dump the decoded $l_{m'}$ bits from the input sequence and repeat from Step 1.

2.9. Symbol Error Probability

When assessing the performance of any error-correcting code, the output error probability is always of interest. In general, error-correcting codes are used to give protection to data which has already been source coded, and hence this error probability is rarely given in terms of the actual source symbols being transmitted, but rather as a bit error probability (assuming the code is binary). Since variable-length codes are used to perform source coding, then in order to judge the performance of VLEC codes we must determine the symbol error probability, i.e. the probability of decoding a transmitted source

symbol in error. Unfortunately, this is not as straight forward to calculate as was the case with fixed-length error-correcting codes. The problem lies in the fact that the number of decoded source symbols may be different to the number sent. This occurs because of the synchronisation losses in variable-length codes, even for the case when the channel does not insert and delete code symbols itself.

2.9.1. Levenshtein Distance

Suppose that the transmitted message is *bacdabd* and that the decoded message is *baaadabd*, where a decoding error has been made with *c* being decoded as *aa*. So clearly in this case, the number of erroneous symbols in the decoded message is two. However, if we were to calculate the Hamming distance between the two strings, as is usually done for fixed-length codes, we encounter two problems. First, the two strings are not of the same length. We can solve this by truncating the longer string to make the two equal in length. However, the Hamming distance still will not give the correct number of errors. For instance, in our example the Hamming distance is five. Clearly, once the number of decoded symbols is not equal to that transmitted, the Hamming distance criterion will give a totally meaningless result.

When dealing with codes capable of correcting deletion and insertion errors, Levenshtein [1965] introduced a new distance measure. We shall use this distance measure in order to define the symbol error probability in the case of VLEC codes. However, first we shall define the Levenshtein distance.

Definition 2.4: Let $\mathbf{a}_1$ and $\mathbf{a}_2$ be two sequences of source symbols over A not necessarily of equal length. Then the *Levenshtein distance* between the two sequences, denoted by $L(\mathbf{a}_1, \mathbf{a}_2)$, is the minimum number of insertions, deletions and substitutions necessary to transform one sequence into the other[11].

It should be appreciated that the Levenshtein distance is not as easily computed as the Hamming distance. Kruskal [1983] gives an algorithm to compute the Levenshtein

[11] Although $L(\mathbf{a}_1, \mathbf{a}_2) = L(\mathbf{a}_2, \mathbf{a}_1)$, we will always take the first listed sequence as the reference sequence and the terms insertion, deletion and substitution are with reference to this sequence.

distance based on the principle of dynamic programming. If the two sequences are of length n, then the complexity of this algorithm is $O(n^2)$. Masek and Paterson [1983] give an improved algorithm for long sequences with complexity $O(n^2/\log n)$. However, both algorithms will take an appreciable time to compute the Levenshtein distance of sequences consisting of a few tens of thousand symbols and hence are not practical to use when message lengths of a few hundred thousand symbols are considered in simulation runs. A more practical algorithm is given in the next section.

Having defined a distance measure between two unequal-length sequences, it is now quite easy to give a definition for the symbol error probability.

Definition 2.5: The *symbol error probability* (SEP) of the decoded source message $\mathbf{a}_r$ when compared to the transmitted source message $\mathbf{a}_t$ is given by the ratio $L(\mathbf{a}_t, \mathbf{a}_r) / |\mathbf{a}_t|$, where $|\mathbf{a}_t|$ denotes the number of source symbols in $\mathbf{a}_t$.

2.9.2. A Practical Algorithm to Evaluate the Symbol Error Probability

In this algorithm we are going to assume that we know the codeword lengths mapped to each source symbol and that the channel does not insert and delete bits (such as the BSC). Both of these assumptions are satisfied for most of the work presented in this thesis.

Let the transmitted source message $\mathbf{a}_t = a_{t_1}a_{t_2}\cdots a_{t_T}$ be encoded using code C as $\mathbf{c}_{t_1}\mathbf{c}_{t_2}\cdots\mathbf{c}_{t_T}$ and let the decoded codeword sequence be $\mathbf{c}_{r_1}\mathbf{c}_{r_2}\cdots\mathbf{c}_{r_R}$ which is mapped to the source message $\mathbf{a}_r = a_{r_1}a_{r_2}\cdots a_{r_R}$. As usual, let $l_i = |\mathbf{c}_i|$, where $\mathbf{c}_i \in C$. Then, the following algorithm will calculate an approximate value for $L(\mathbf{a}_t, \mathbf{a}_r)$.

1. Let $i, j = 1$ and $Ld = 0$.
2. If $i > T$, then increment Ld by $R-j+1$ and stop.
3. If $j > R$, then increment Ld by $T-i+1$ and stop.
4. If $a_{t_i} = a_{r_j}$ then increment i and j and repeat from Step 2, otherwise go to Step 5.
5. If $|\mathbf{c}_{t_i}| = |\mathbf{c}_{r_j}|$ then increment i, j and Ld and go back to Step 2, otherwise proceed to Step 6.

6. Let $\mathbf{a}_1 = a_{t_i}a_{t_{i+1}}\cdots a_{t_{i+u}}$ and $\mathbf{a}_2 = a_{r_j}a_{r_{j+1}}\cdots a_{r_{j+v}}$ such that u and v are the smallest integers which satisfy $|\mathbf{c}_{t_i}\mathbf{c}_{t_{i+1}}\cdots\mathbf{c}_{t_{i+u}}| = |\mathbf{c}_{r_j}\mathbf{c}_{r_{j+1}}\cdots\mathbf{c}_{r_{j+v}}|$ or $i+u = T$ and $j+v = R$. Increment Ld by $L(\mathbf{a}_1, \mathbf{a}_2)$ and i, j by $u+1$ and $v+1$ respectively. Go back to Step 2.

At the end of this algorithm, $L(\mathbf{a}_t, \mathbf{a}_r) \approx Ld$. However, note that in any case $Ld \geq L(\mathbf{a}_t, \mathbf{a}_r)$. This is a direct consequence of Definition 2.4.

The algorithm given above is much faster, since essentially it is evaluating the Hamming distance between those parts of the sequences which are in synchronisation (since here there are no insertions or deletions of symbols) and only uses the Levenshtein distance between those parts of the sequences which are out of synchronisation. In general, these tend to be relatively short sequences and hence the complexity of calculating the Levenshtein distance in this case is much reduced.

Having obtained an approximate value for $L(\mathbf{a}_t, \mathbf{a}_r)$, then an approximate value for the symbol error probability is easily obtained. However, we should point out that it seems to us that this method reflects better what is really happening in practice and hence is a more realistic value for the symbol error probability. Having said this, it seems appropriate to point out that using simulation results we were able to confirm that in practice the difference between the approximate method and the exact method is negligible for all codes considered.

2.10. Synchronisation-Error-Correcting Codes

Another class of codes which was considered in the literature is that of synchronisation-error-correcting codes, i.e. codes which are capable of correcting insertion and deletions (and possibly substitution) errors. As may be appreciated, these types of errors are much more difficult to correct then just simple substitution errors. Consequently, all the codes designed to combat these types of errors are, to our knowledge, of fixed-length.

Sellers [1962] was the first to consider this type of codes and gave a construction for fixed-length codes capable of correcting single bit synchronisation errors. His code was constructed by inserting special symbols into a burst error-correcting code. Ullman [1966]

also gives a single synchronisation-error-correcting code. Levenshtein [1965a, 1965b, 1966] investigates some properties of synchronisation-error-correcting codes and proposed a binary code capable of correcting t deletions or insertions of ones. Ullman [1966] gives upper and lower bounds on the redundancy required to correct synchronisation errors. A family of codes which can correct some substitution and synchronisation errors is given by Calabi and Hartnett [1969a]. The authors also give some further characterisations for these codes [Calabi & Hartnett, 1969b]. Tanaka and Kasai [1976] derive some sufficient conditions, based on a generalised Levenshtein distance, for codes to correct synchronisation errors and a method for constructing such codes. Recently, Hollmann [1993] gives some further characterisations by separating the insertion and deletion errors for synchronisation-error-correcting codes.

2.11. Conclusion

We have seen that variable-length codes are more difficult to deal with than fixed-length codes even for the noiseless case. Here, all that is required for a fixed-length code to be useful is that it be non-singular. Variable-length codes, however, need also to be uniquely decodable. To minimise the decoding delay, they must also be instantaneously decodable.

For channels which admit insertion and deletion errors, we have seen that we need to design into both fixed and variable-length codes special characteristics which will enable the decoder to recover synchronisation. In the case of variable-length codes, these synchronisation problems are present even with only substitution errors, which in practice are more common. Hence, for variable-length codes used over a noisy channel there must always be some mechanism which enables the decoder to re-synchronise. This problem can be minimised if the variable-length code is protected by using an error-correcting code, which effectively would yield a noiseless channel for the variable-length code. Statistically synchronisable variable-length codes may also offer good performance on a noisy channel. The property of self-synchronisation in some variable-length codes is attractive even for

channels with insertion and deletion errors, since no further control logic is required to maintain synchronisation unlike for the case of fixed-length codes.

Finally, we have introduced variable-length error-correcting codes, which have in-build additional structure to correct substitution errors, and hence improving also their synchronisation properties. A construction algorithm for perfect α-prompt codes was given which could also be used to construct other α-prompt codes. Three instantaneous decoding algorithms for the class of α-prompt codes were also given. All three of them, with the possible exception of the tail decoding algorithm, consider the VLEC codes as pseudo block codes. This limits the performance of these codes as we shall see in the next chapter. Note that a performance comparison for these algorithms is deferred until Chapter 3, where a maximum likelihood decoding algorithm for VLEC codes is derived.

Chapter 3.

Trellis Structure of Variable-Length Error-Correcting Codes

3.1. Introduction

In this chapter we present a novel way of decoding VLEC codes by treating them as trellis codes. Consequently it is shown that in many respects they behave very much like convolutional codes and exhibit a form of "memory", although in the case of VLEC codes the memory is not related to an encoding shift register, but is due to the different codeword lengths.

Using this representation, we give a maximum likelihood decoding algorithm based on the Viterbi algorithm and also derive a maximum a-posteriori (MAP) metric for use with this algorithm.

There are other properties of VLEC codes similar to those of convolutional codes, like their free distance, constraint length and catastrophic behaviour, which are dealt with here.

In this chapter we also give two methods, namely using union bounds and computer simulation, to determine the performance of these codes on the BSC channel, and we compare these two methods. We also compare the performance of VLEC codes with maximum likelihood and MAP decoding. Finally, we do comparisons with the decoding algorithms presented in Chapter 2 and show that maximum likelihood decoding of VLEC codes achieves an appreciable coding gain over the instantaneous algorithms.

The tree and trellis representations for VLEC codes appeared first in [Buttigieg & Farrell, 1993a], whereas the maximum a-posteriori metric for VLEC code was first derived in [Buttigieg & Farrell, 1993b]. Comparisons between maximum likelihood decoding for VLEC codes and instantaneous decoding were presented in [Buttigieg & Farrell, 1994b].

3.2. Maximum Likelihood Decoding

Let C be a binary VLEC code ($s_1@L_1$, $s_2@L_2$, $\cdots$, $s_\sigma@L_\sigma$) as defined in Section 2.2. Under noisy conditions, the decoder for C has two main problems. First to determine the codeword boundaries, and secondly to determine the codeword values. These two problems must be solved simultaneously in a maximum likelihood decoder. Hence, if we are to implement a maximum likelihood decoder, we must use a representation for VLEC codes which retains both the spatial and amplitude information. This can be achieved by using a tree structure.

3.2.1. Tree Structure

In this representation, the root node of the tree represents the start of the message. Each node in the tree is connected to s other nodes. The s branches connecting these nodes are each labelled with a different codeword of C.

Definition 3.1: A *path* p_i through the tree is any sequence of branches (codewords) connecting the root node to some other node i.

Definition 3.2: The *span* of the path p_i, denoted by $\|p_i\|$, is the number of branches (codewords) in the path. The *length* of the path p_i, denoted by $|p_i|$, is the total number of bits in the path.

The nodes in the tree are organised in such a way that all nodes with equal path lengths fall on the same vertical line (assuming the tree grows horizontally). The first tree segment for C is shown in Figure 3.1. This is also the basic building block for the tree, because this structure is repeated at each new node generated.

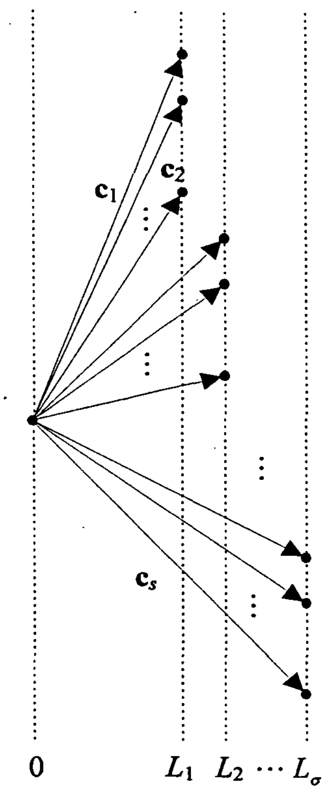

Figure 3.1: First tree segment for a VLEC code C

Example 3.1: Consider the VLEC code C_4 given in Table 3.1. For this code $\sigma = 2$, $s = 3$, $s_1 = 1$, $s_2 = 2$ and $L_1 = 3$, $L_2 = 4$. The tree diagram for this code, up to a maximum path length of 12 bits, is shown in Figure 3.2.

Source Symbol	Codeword
a	000
b	0110
c	1011

Table 3.1: VLEC Code C_4

Now, assume that the channel does not insert or delete channel symbols (bits), such as the BSC for the binary case. Then, knowing the number of received bits N, the decoder can determine the set of possible paths $P_N = \{p_i : |p_i| = N\}$ through the tree, which corresponds to all the nodes on the same vertical line at bit position N. Then, with

probability one, the transmitted path, $p_m \in P_N$. Hence, a maximum likelihood decoder need only consider the sub-set P_N of all the paths in the tree.

Let the codeword sequence associated with the path $p_i \in P_N$ be $(\mathbf{c}_{i_1}, \mathbf{c}_{i_2}, \cdots, \mathbf{c}_{i_{\eta_i}})$ and let $\mathbf{f}_i = \mathbf{c}_{i_1}\mathbf{c}_{i_2}\cdots\mathbf{c}_{i_{\eta_i}}$, where $\eta_i = \|p_i\|$. Then, by definition, $|\mathbf{f}_i| = N$. Hence, the set

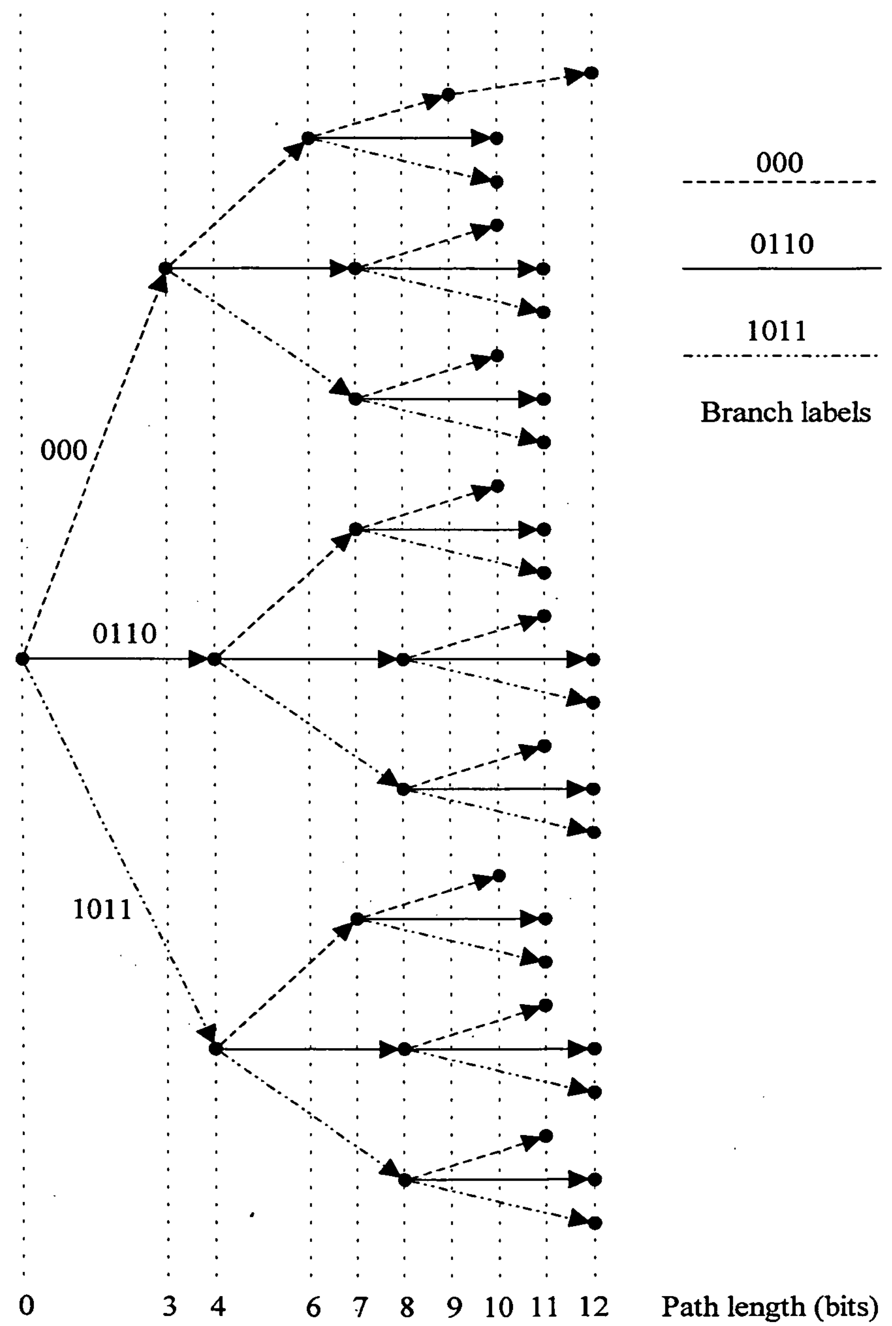

Figure 3.2: Tree diagram for code C_4 up to length 12 bits

$F_N = \{\mathbf{f}_i : |\mathbf{f}_i| = N\}$ has a one-to-one mapping to P_N given by $p_i \leftrightarrow \mathbf{f}_i$.

Definition 3.3: The set $F_N = \{\mathbf{f}_i : |\mathbf{f}_i| = N\}$, $\mathbf{f}_i = \mathbf{c}_{i_1}\mathbf{c}_{i_2}\cdots\mathbf{c}_{i_{\eta_i}}$, $\mathbf{c}_{i_j} \in C\ \forall\, j = 1, 2, \cdots, \eta_i$, is the *extended code* of the VLEC code C of order N.

Note that the extended code F_N is a fixed-length code with codeword length equal to N. Assuming that all codewords of F_N are equally probable (but see Section 3.3 below), it can easily be proved that for maximum likelihood decoding of F_N over the BSC, we need to choose that codeword which is at the minimum Hamming distance to the received bit sequence [Sklar, 1988]. Therefore, since the transmitted sequence of codewords is in F_N with probability one, then by performing maximum likelihood decoding on F_N we would also be achieving maximum likelihood decoding of the associated VLEC code C.

3.2.2. Trellis Structure

In the previous section we have derived a maximum likelihood decoding algorithm for VLEC codes using the tree structure. However, one problem with this structure is that it grows exponentially with increasing N. To overcome this problem, we reduce the tree structure to a trellis structure in a similar manner as is done for convolutional codes [Clark & Cain, 1981]. We can achieve this by combining all nodes on the same vertical line (i.e. all the nodes with paths of the same length) into one state in the trellis. We shall prove in Section 3.2.2.2 that the trellis structure so constructed can still be used to perform maximum likelihood decoding of VLEC codes.

Definition 3.4: Let S_i represent the state in the trellis diagram for the VLEC code C corresponding to bit position i within the encoded message. Then, a *path* p_i^j in the trellis is a particular choice of transitions to traverse from state S_0 to some other state S_i, where j is an arbitrary path index, since there may be more than one path to state S_i.

It should be obvious from the above definition that $|p_i^j| = i$. Note that a path may be characterised in two equivalent ways. It could either be given as a sequence of source symbols, which would constitute the source message, or equivalently, it could be given as a sequence of codewords, which would constitute the encoded message.

Definition 3.5: A path $p_i^j = (\mathbf{c}_{k_1}, \mathbf{c}_{k_2}, \cdots, \mathbf{c}_{k_{\eta_k}})$ is *extended* into s new paths, $p_{i+l_1}^{j_1}$, $p_{i+l_2}^{j_2}$, $\cdots$, $p_{i+l_s}^{j_s}$, by adjoining to it all possible codewords in the code C, i.e. $p_{i+l_1}^{j_1} = (\mathbf{c}_{k_1}, \mathbf{c}_{k_2}, \cdots, \mathbf{c}_{k_{\eta_k}}, \mathbf{c}_1)$, $p_{i+l_2}^{j_2} = (\mathbf{c}_{k_1}, \mathbf{c}_{k_2}, \cdots, \mathbf{c}_{k_{\eta_k}}, \mathbf{c}_2)$, $\cdots$, $p_{i+l_s}^{j_s} = (\mathbf{c}_{k_1}, \mathbf{c}_{k_2}, \cdots, \mathbf{c}_{k_{\eta_k}}, \mathbf{c}_s)$.

Hence the number of transitions[1] (source symbols or codewords) in the extended paths is one more than that in the original path. There are σ new distinct destination states in the extended paths, S_{i+L_1}, S_{i+L_2}, $\cdots$, S_{i+L_σ}, respectively.

Definition 3.6: Two states S_i and S_j, with $i < j$, are said to be *consecutive* iff there is no state S_k such that $i < k < j$.

3.2.2.1. Trellis Construction Algorithm

1. Let S_0 be the first state in the trellis.
2. From each existing state S_i emit transitions for each of the codewords in C with destination states S_{i+L_1}, S_{i+L_2}, $\cdots$, S_{i+L_σ}. If any of these states do not already exist, then these are created, otherwise the transitions are made to existing states.
3. The states are grouped into stages, each stage consisting of consecutive states starting with S_{jL_σ}, $j = 0, 1, 2, \cdots$, where a *stage* of the trellis is defined to be any vertical group of states within the trellis.

Note that step 3 is not essential for the decoding algorithm as will be given in Section 3.2.2.2 and is only required to force transitions to be either within the same stage or from the previous stage only. In this case, the number of states in a stage is directly related to the constraint length of the VLEC code, as we shall see in Section 3.4.2.

Example 3.2: Again, consider code C_4 given in Table 3.1. The trellis diagram for this code is shown in Figure 3.3 up to bit position 15. Notice that same length codewords give rise to parallel transitions. Compare this with the equivalent tree diagram of Figure 3.2. Whereas the trellis starts to repeat itself after state S_9, the tree diagram continues to grow exponentially.

[1] Note that the terms 'transition' and 'branch' will be used interchangeably throughout this text.

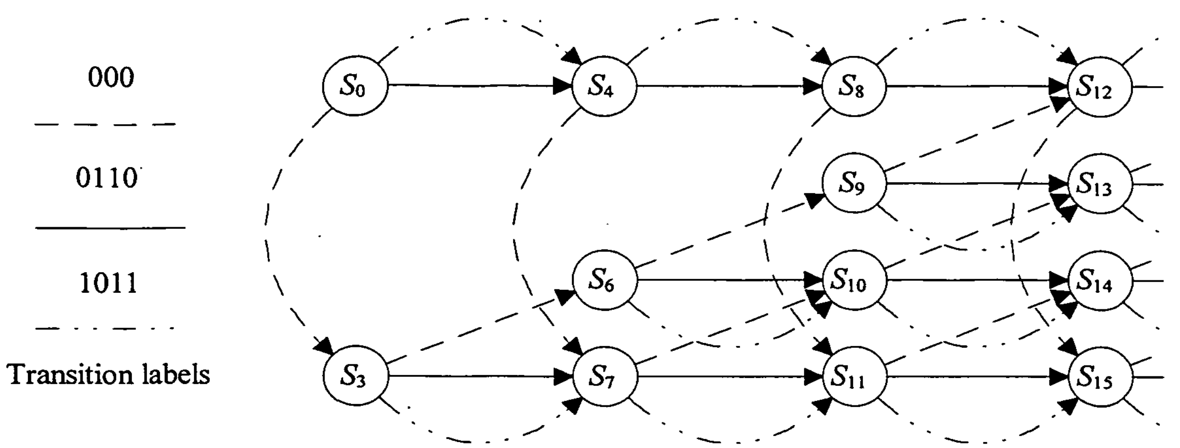

Figure 3.3: Trellis diagram for code C_4

Notice that the states in Figure 3.3 are arranged in such a way that the transitions are either within the same stage or from the previous stage only, as enforced by step 3 in the construction algorithm.

An alternative trellis diagram for code C_4 is shown in Figure 3.4. Here, the fact that each state in the trellis is identical is being emphasised. Note that Figures 3.3 and 3.4 are equivalent and their only difference lies in the way in which the states are laid out.

3.2.2.2. Modified Viterbi Algorithm

We shall now give a modified version of the Viterbi decoding algorithm [Viterbi, 1967] for VLEC codes. A decoding algorithm for variable-length trellises, as used for run-length codes, is also discussed by Belongie and Heegard [1993].

Let $\mathbf{y} = y_1y_2 \cdots y_N$ be the received N-bit sequence and denote the metric of the surviving path at state S_i by M_i.

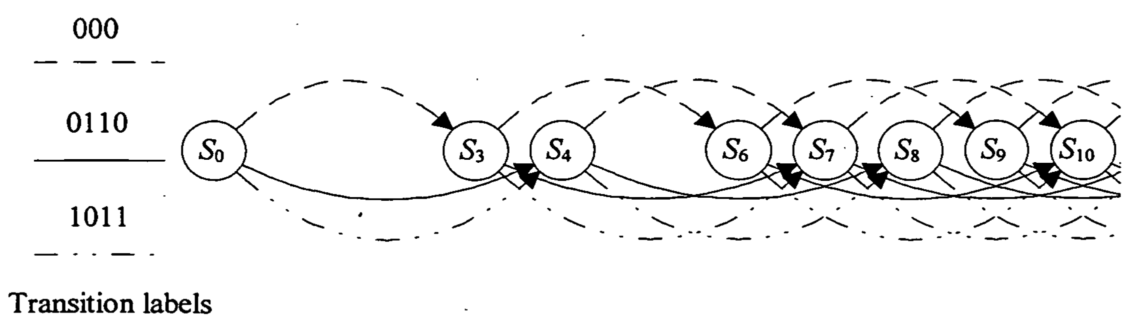

Figure 3.4: Alternative trellis diagram for code C_4

1. Assign $M_0 = 0$ and $M_i = -\infty$, $\forall\ i > 0$. Let S_i denote the current state and initially put $i = 0$.
2. For all codewords $\mathbf{c}_j \in C$ evaluate the branch metric $m_j = -H(\mathbf{c}_j, y_i y_{i+1} \cdots y_{i+l_j-1})$. Flag S_{i+l_j} as a visited state. If $m_j + M_i > M_{i+l_j}$ then store this codeword for the transition $S_i \rightarrow S_{i+l_j}$ (overwriting any other previously stored transitions to state S_{i+l_j}) and make $M_{i+l_j} = m_j + M_i$.
3. Increment i to the next visited state and repeat step 2 until $i > N - l_1$.
4. Decode the message corresponding to the codeword sequence represented by the surviving path to state S_N.

Theorem 3.1: The modified Viterbi decoding algorithm given above achieves maximum likelihood decoding for VLEC codes (assuming all paths are equally probable; but see Section 3.3 below).

Proof: In step 4 of the algorithm above, we are decoding the surviving sequence of codewords to state S_N. Hence, the number of bits in the decoded codeword sequence is equal to N and therefore the decoded sequence is a codeword of F_N. Now we need to prove that it is also that sequence with the minimum Hamming distance to $\mathbf{y}$.

Suppose that the minimum distance path to state S_N, p_N^m is eliminated at some previous state S_j, as shown in Figure 3.5. This implies that the distance between the received bit sequence and p_N^m at state S_j is greater than that for the surviving path[2]. Now, consider that we follow the surviving path at state S_j with the remaining part of p_N^m from state S_j to state S_N. Then, in this case, the distance for the complete surviving path will be less than that for the minimum distance path, p_N^m, which is a contradiction. Hence, the surviving path at state S_N must be the one with minimum distance to the received bit sequence (i.e. the one with maximum metric) and this is the maximum likelihood path. ■

[2] Notice that due to the negative sign in front of the Hamming distance in step 2, the maximum metric path is the path with the minimum distance.

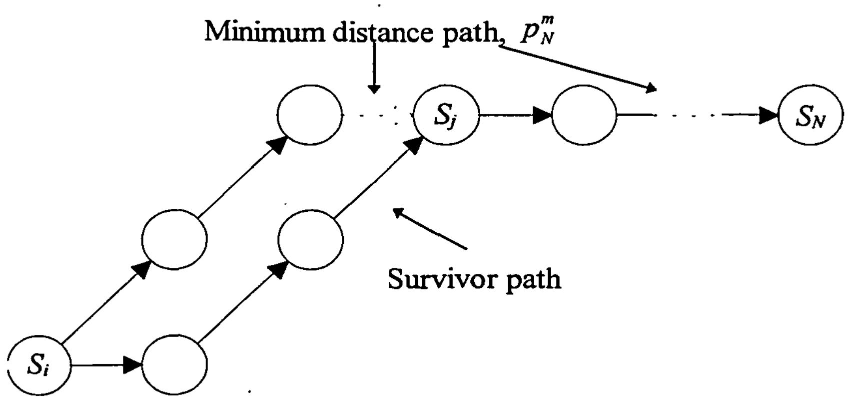

Figure 3.5: Proof of maximum likelihood decoding

3.3. Maximum A-Posteriori Metric

In Section 3.2 we assumed that all the paths to state S_N were equally probable. In practice this is not true. Even if the codewords of C occur with equal probabilities, which again is in general not true for VLEC codes, the paths to state S_N may have different probabilities. The reason for this is that these paths may have a different number of codewords. In Theorem 3.2 we drop this assumption to obtain a maximum a-posteriori (MAP) metric for VLEC codes.

Theorem 3.2: For MAP decoding of VLEC codes, the branch metric m_j between states S_i and S_{i+l_j} used in the modified Viterbi algorithm is given by

$$m_j = [\log p - \log(1-p)]H\left(\mathbf{c}_j, y_i y_{i+1} \cdots y_{i+l_j-1}\right) + \log P\left(\mathbf{c}_j\right)$$

where, p is the cross-over probability of the BSC, $H\left(\mathbf{c}_j, y_i y_{i+1} \cdots y_{i+l_j-1}\right)$ is the Hamming distance between codeword $\mathbf{c}_j$ and the corresponding received bits starting at bit position i, and $P(\mathbf{c}_j)$ is the probability of occurrence of $\mathbf{c}_j$.

Proof: Let $\mathbf{y}$ denote the received N-bit encoded message and $\mathbf{f}_i \in F_N$, $\forall\ i = 1, 2, \cdots, f_N$, where F_N is the extended code of order N of the VLEC code C and f_N is the cardinality of F_N. For MAP decoding, we need to choose that codeword $\mathbf{f}_m$ such that

$$P(\mathbf{f}_m \mid \mathbf{y}) \geq P(\mathbf{f}_i \mid \mathbf{y}) \tag{3.1}$$

$\forall\ i = 1, 2, \cdots, f_N$ and $1 \leq m \leq f_N$.

Now, for the BSC, we have

$$P(\mathbf{y}|\mathbf{f}_i) = p^{h_i}(1-p)^{N-h_i} \tag{3.2}$$

where, p is the cross-over probability of the BSC and $h_i = H(\mathbf{y}, \mathbf{f}_i)$.

Also, using Bayes' theorem [Papoulis, 1965], we have

$$\begin{aligned} P(\mathbf{f}_m|\mathbf{y}) &= \frac{P(\mathbf{y}|\mathbf{f}_m)P(\mathbf{f}_m)}{P(\mathbf{y})} \\ &= \frac{P(\mathbf{y}|\mathbf{f}_m)P(\mathbf{f}_m)}{\sum_{i=1}^{f_N} P(\mathbf{y}|\mathbf{f}_i)P(\mathbf{f}_i)} \end{aligned} \tag{3.3}$$

where, $P(\mathbf{f}_i)$ is the probability that $\mathbf{f}_i$ is transmitted.

Substituting for $P(\mathbf{y}|\mathbf{f}_i)$ and $P(\mathbf{y}|\mathbf{f}_m)$ in equation (3.3), we have

$$P(\mathbf{f}_m|\mathbf{y}) = \frac{p^{h_m}(1-p)^{N-h_m}P(\mathbf{f}_m)}{\sum_{i=1}^{f_N} p^{h_i}(1-p)^{N-h_i}P(\mathbf{f}_i)}. \tag{3.4}$$

Hence, by expression (3.1), for MAP decoding we need to choose $\mathbf{f}_m$ so as to maximise equation (3.4). Now, the denominator of (3.4) is constant for all $\mathbf{f}_m$. Hence, to maximise $P(\mathbf{f}_m|\mathbf{y})$, we only need maximise the numerator of (3.4). Instead of maximising $P(\mathbf{f}_m|\mathbf{y})$, it is more convenient to maximise its logarithm. This can be done since the logarithmic function is monotonically increasing. Therefore, we need to choose $\mathbf{f}_m$ to maximise

$$\log\left[p^{h_m}(1-p)^{N-h_m}P(\mathbf{f}_m)\right] = h_m\left[\log p - \log(1-p)\right] + N\log(1-p) + \log P(\mathbf{f}_m). \tag{3.5}$$

In addition, since $N\log(1-p)$ is the same for all $\mathbf{f}_m$, then we only need to choose $\mathbf{f}_m$ so as to maximise

$$h_m\left[\log p - \log(1-p)\right] + \log P(\mathbf{f}_m). \tag{3.6}$$

The metric given in expression (3.6) is for complete paths to state S_N. In order to apply this metric in the modified Viterbi decoding algorithm as given in Section 3.2.2.2, we need to modify it for single transitions in the trellis, which represent single codewords.

Let $\mathbf{f}_i$ be made up of η_i codewords of C, i.e. $\mathbf{f}_i = \mathbf{c}_{i_1}\mathbf{c}_{i_2}\cdots\mathbf{c}_{i_{\eta_i}}$, where $\mathbf{c}_{i_j}$ is the jth codeword of the ith path to state S_N. Define the decomposition of $\mathbf{y}$ with respect to $\mathbf{f}_i$ to be $\mathbf{y} = \mathbf{y}_{i_1}\mathbf{y}_{i_2}\cdots\mathbf{y}_{i_{\eta_i}}$, where $|\mathbf{y}_{i_j}| = |\mathbf{c}_{i_j}|$ for all $j = 1, 2, \cdots, \eta_i$. Note that $\mathbf{y}_{i_j}$ represents that part of

the received bit sequence which is compared with the jth transition of the ith path to state S_N. It is very easy to see that

$$h_i = \sum_{j=1}^{\eta_i} H\left(\mathbf{y}_{i_j}, \mathbf{c}_{i_j}\right) \tag{3.7}$$

Also, assuming that the source is memory-less, then the probability of $\mathbf{f}_i$, $P(\mathbf{f}_i)$, is given by

$$\begin{aligned} P(\mathbf{f}_i) &= P\left(\mathbf{c}_{i_1}\right)P\left(\mathbf{c}_{i_2}\right)\cdots P\left(\mathbf{c}_{i_m}\right) \\ \Rightarrow \log P(\mathbf{f}_i) &= \sum_{j=1}^{\eta_i} \log P\left(\mathbf{c}_{i_j}\right). \end{aligned} \tag{3.8}$$

Therefore, substituting for h_m and $P(\mathbf{f}_m)$ in the metric given by expression (3.6), for MAP decoding we need to maximise

$$[\log p - \log(1-p)]\sum_{j=1}^{\eta_m} H\left(\mathbf{y}_{m_j}, \mathbf{c}_{m_j}\right) + \sum_{j=1}^{\eta_m} \log P\left(\mathbf{c}_{m_j}\right). \tag{3.9}$$

Hence, we need to choose at each state, that codeword $\mathbf{c}_{m_j}$ which maximises the running sum of expression (3.10).

$$[\log p - \log(1-p)]H\left(\mathbf{y}_{m_j}, \mathbf{c}_{m_j}\right) + \log P\left(\mathbf{c}_{m_j}\right) \tag{3.10}$$

Therefore, for MAP decoding the branch metric m_j used in the modified Viterbi algorithm given in Section 3.2.2.2 (step 2) is modified to

$$m_j = [\log p - \log(1-p)]H\left(\mathbf{c}_j, y_{i+1}y_{i+2}\cdots y_{i+l_j-1}\right) + \log P\left(\mathbf{c}_j\right) \tag{3.11}$$

■

If $P(\mathbf{c}_i) = P(\mathbf{c}_j)$ and $|\mathbf{c}_i| = |\mathbf{c}_j|$, $\forall\ \mathbf{c}_i, \mathbf{c}_j \in C$, then the MAP metric given by equation (3.11) reduces to $m_j = [\log p - \log(1-p)]\ H(\mathbf{c}_j, y_i y_{i+1}\cdots y_{i+l_j-1})$ since in this case the term $\sum_{j=1}^{\eta_m} \log P\left(\mathbf{c}_{m_j}\right)$ in expression (3.9) will be a constant and hence may be ignored. If we assume that $0 \le p < 0.5$ [3], then $[\log p - \log(1-p)] < 0$. Hence in this case $m_j = -\kappa H(\mathbf{c}_j, y_i y_{i+1}\cdots y_{i+l_j-1})$ where κ is a positive constant for a given p. Since κ is a constant for all codewords, then it can be ignored and we have $m_j = -H(\mathbf{c}_j, y_i y_{i+1}\cdots y_{i+l_j-1})$. This is the same as the maximum likelihood metric given in Section 3.2.2.2 since the two conditions given above ensure that all paths through the trellis to state S_N are equally probable.

[3] We can always assume that p is within this range, since if it is greater than 0.5 then we can simple invert the 0's and 1's at the output of the BSC.

3.4. Some Properties of VLEC codes

In the previous sections we have established that we can consider VLEC codes to be trellis codes, their "memory" arising not from some storage element, but from the spatial information. Consequently, the properties of VLEC codes should be similar to those of trellis codes or convolutional codes, which are a special class of time-invariant linear trellis codes.

3.4.1. Free Distance

One of the most important parameters for convolutional codes when using maximum likelihood decoding is the free distance. As we shall see in Section 3.5, this is also an important parameter for VLEC codes which determines the performance for the code.

Definition 3.7: The *minimum block distance for length L_k, b_k,* of a VLEC code C is defined as the minimum Hamming distance between all codewords with the same length L_k, i.e.

$$b_k = \min\{H(\mathbf{c}_i, \mathbf{c}_j) : \mathbf{c}_i, \mathbf{c}_j \in C, i \neq j \text{ and } |\mathbf{c}_i| = |\mathbf{c}_j| = L_k\} \tag{3.12}$$

There are σ different minimum block distances, one for each different codeword length. However, if for some length L_k there is only one codeword, i.e. $s_k = 1$, then in this case the minimum block distance for length L_k is undefined.

Definition 3.8: The *overall minimum block distance*, $b_{\min}$, of a VLEC code C is defined as the minimum value of b_k over all $k = 1, 2, \cdots, \sigma$.

Definition 3.9: The *diverging distance* between two codewords $\mathbf{c}_i = c_{i_1}c_{i_2}\cdots c_{i_{l_i}}$ and $\mathbf{c}_j = c_{j_1}c_{j_2}\cdots c_{j_{l_j}}$, $D(\mathbf{c}_i, \mathbf{c}_j)$, where $\mathbf{c}_i, \mathbf{c}_j \in C$, $l_i = |\mathbf{c}_i|$ and $l_j = |\mathbf{c}_j|$, with $l_i > l_j$, is defined as

$$D(\mathbf{c}_i, \mathbf{c}_j) = H(c_{i_1}c_{i_2}\cdots c_{i_{l_j}}, c_{j_1}c_{j_2}\cdots c_{j_{l_j}}). \tag{3.13}$$

Note that $D(\mathbf{c}_i, \mathbf{c}_j) = D(\mathbf{c}_j, \mathbf{c}_i)$.

The *minimum diverging distance* of the VLEC C, $d_{\min}$, is the minimum value of all the diverging distances between all possible pairs of unequal length codewords of C, i.e.

$$d_{\min} = \min\{D(\mathbf{c}_i, \mathbf{c}_j) : \mathbf{c}_i, \mathbf{c}_j \in C, |\mathbf{c}_i| \neq |\mathbf{c}_j|\} \tag{3.14}$$

Definition 3.10: The *converging distance* between two codewords $\mathbf{c}_i = c_{i_1}c_{i_2}\cdots c_{i_{l_i}}$ and $\mathbf{c}_j = c_{j_1}c_{j_2}\cdots c_{j_{l_j}}$, $C(\mathbf{c}_i, \mathbf{c}_j)$, where $\mathbf{c}_i, \mathbf{c}_j \in C$, $l_i = |\mathbf{c}_i|$ and $l_j = |\mathbf{c}_j|$, with $l_i > l_j$, is defined as

$$C(\mathbf{c}_i, \mathbf{c}_j) = H(c_{i_{l_i-l_j+1}}c_{i_{l_i-l_j+2}}\cdots c_{i_{l_i}}, c_{j_1}c_{j_2}\cdots c_{j_{l_j}}). \tag{3.15}$$

Again, note that $C(\mathbf{c}_i, \mathbf{c}_j) = C(\mathbf{c}_j, \mathbf{c}_i)$.

The *minimum converging distance* of the VLEC C, $c_{\min}$, is the minimum value of all the converging distances between all possible pairs of unequal length codewords of C, i.e.

$$c_{\min} = \min \{C(\mathbf{c}_i, \mathbf{c}_j) : \mathbf{c}_i, \mathbf{c}_j \in C, |\mathbf{c}_i| \neq |\mathbf{c}_j|\} \tag{3.16}$$

A more complete description of a VLEC code may now be given by extending the notation introduced in Section 2.3. A VLEC code C ($s_1@L_1$, b_1; $s_2@L_2$, b_2; $\cdots$; $s_\sigma@L_\sigma$, b_σ; $d_{\min}$, $c_{\min}$) is one which has s_i codewords of length L_i with minimum block distance for length L_i, b_i for all $i = 1, 2, \cdots, \sigma$, where σ is the different number of codeword lengths, and a minimum diverging distance $d_{\min}$ and minimum converging distance $c_{\min}$. An undefined minimum block distance for some length is denoted by a '-'.

Definition 3.11: The *free distance*, d_{free} of a VLEC code C is defined to be the minimum Hamming distance in the set of all arbitrary long paths that diverge from some common state S_i and converge again in another common state $S_j, j > i$. More formally

$$d_{\text{free}} = \min \{H(\mathbf{f}_i, \mathbf{f}_j) : \mathbf{f}_i, \mathbf{f}_j \in F_N, N = 1, 2, \cdots, \infty\} \tag{3.17}$$

where, F_N is the extended code of order N of C [4].

Theorem 3.3: The free distance of a VLEC code C is bounded by

$$d_{\text{free}} \geq \min(b_{\min}, d_{\min} + c_{\min}) \tag{3.18}$$

Proof: Following Definition 3.11 for the free distance, there are two cases to consider (see Figure 3.6).

Case 1: If the two transitions emanating from the first state S_i are due to codewords of the same length, then these may give rise to two different paths from state S_i to S_j arising from two parallel transitions from state S_i to some state S_k and the same transitions from state S_k to the final state S_j. Hence the free distance in this case will be equal to the overall minimum block distance $b_{\min}$.

[4] Note that for certain values of N the set F_N may be empty.

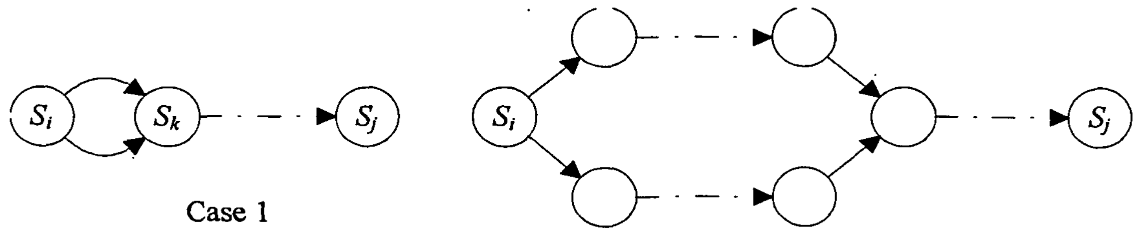

Figure 3.6: Proof of Theorem 3.3

Case 2: If the transitions emanating from the first state S_i are due to codewords of differing lengths, then the next states must be different. The distance between the two paths must therefore be at least equal to the diverging distance between the two codewords. However, at some point the two paths must re-converge to some common state. Therefore, the distance between the two paths is increased by the converging distance. Hence the minimum distance for Case 2 is at least $d_{\min}+c_{\min}$.

The theorem follows by combining the two cases. ■

3.4.2. Constraint Length

Another important parameter in convolutional codes is the constraint length [Lin & Costello, 1983]. There are several different definitions for the constraint length in the case of convolutional codes, but the one that seems most adaptable to VLEC codes is the one related to the number of stages in the shift registers for the convolutional code [Viterbi, 1971]. If we call this number K, then the number of states in the state diagram for the convolutional code will be 2^K.

Similarly, we may define a constraint length for VLEC codes based on the "memory" of the VLEC code.

Definition 3.12: Let μ be the minimum number of consecutive states in the repetitive part of the trellis diagram for a VLEC code C required to build up all subsequent states. Then, the *constraint length*, K, of C is defined to be

$$K \triangleq \log_2 \mu. \tag{3.19}$$

Using this definition, μ is the number of states in any stage of the trellis for the VLEC code if this is constructed using the algorithm given in Section 3.2.2.1.

The following lemma is useful to prove Theorem 3.4, which gives a relationship between the constraint length of a VLEC code C and the lengths of its codewords.

Lemma 3.1: In the repetitive part of the trellis diagram of a VLEC code C, two states S_i and S_j, $j > i$, are consecutive iff $j = i + g$, where g is the greatest common divisor (gcd) of all the different codeword lengths in C.

Proof: Let N be the number of bits in a path. Then N is given by

$$N = n_1L_1 + n_2L_2 + \cdots + n_\sigma L_\sigma \tag{3.20}$$

where, n_i is the number of codewords in the path with length L_i, $i = 1, 2, \cdots, \sigma$. Let $g = \gcd(L_1, L_2, \cdots, L_\sigma)$, then there exist positive integers κ_i such that $L_i = \kappa_i g$, $\forall\ i = 1, 2, \cdots, \sigma$. Therefore, we can write equation (3.20) as

$$N = \sum_{i=1}^{\sigma} n_i \kappa_i g = g \sum_{i=1}^{\sigma} n_i \kappa_i \,. \tag{3.21}$$

Hence, N must be a multiple of g. Therefore, for sufficiently large N (when the trellis starts to repeat itself), the difference between two consecutive states is g bits. ■

Theorem 3.4: The constraint length, K, of a VLEC code C with codeword lengths L_1, $L_2, \cdots, L_\sigma$ with $L_1 < L_2 < \cdots < L_\sigma$, is given by

$$K = \log_2\left(\frac{L_\sigma}{\gcd(L_1, L_2, \cdots, L_\sigma)}\right) \tag{3.22}$$

Proof: The state S_i representing bit position i is built up from the transitions arising from the states $S_{i-L_1}, S_{i-L_2}, \cdots, S_{i-L_\sigma}$. Therefore, the most distant state is S_{i-L_σ}. Hence, at most, we need to store L_σ states to build the future states. But, by Lemma 3.1, the difference between the bit positions represented by any two consecutive states is given by $g = \gcd(L_1, L_2, \cdots, L_\sigma)$. Hence, the minimum number of states required to build up all subsequent states in the repetitive part of the trellis, μ, is L_σ / g. But, by definition, $K = \log_2 \mu$. Hence the theorem is proved. ■

3.4.3. Catastrophic Codes

As with convolutional codes, VLEC codes may exhibit catastrophic behaviour if not properly designed. A VLEC code is said to be catastrophic if a finite number of errors on the channel may cause an infinite number of decoded symbol errors.

Example 3.3: Consider code C_5 (1@4,-; 1@5,-; 1@6,-; 2,2) given in Table 3.2. If the message *ccc*... is transmitted and there are three bit errors, in the first, fourth and fifth bit positions, then the decoder will never re-synchronise and the output will be *baaa*..., as shown in Figure 3.7. Hence, in this case, three bit errors will cause an infinite number of symbol errors and thus C_5 is catastrophic.

Source Symbol	Codeword
a	0101
b	00110
c	101010

Table 3.2: An example for a catastrophic VLEC code C_5

Transmitted message:	*c*	*c*	*c*		...
Encoded message:	1 0 1 0 1 0	1 0 1 0 1 0	1 0 1 0 1 0		...
Received message:	**0** 0 1 **1 0**	0 1 0 1	0 1 0 1	0 1 0 1	0 ...
Decoded message:	*b*	*a*	*a*	*a*	...

Figure 3.7: Catastrophic behaviour of C_5

One may notice that the above problem is similar to the problem of bounded synchronisation delay encountered in Chapter 2. In fact, it is obvious that if a VLEC code is catastrophic, then the code does not have a finite synchronisation delay as defined in Definition 2.1 on page 39. However, the converse is not true. Consider the VLEC code C_6 (1@2,-; 1@3,-; 2, 2) given in Table 3.3. This code does not have a finite synchronisation delay since if the message *aaa*... is transmitted and the first bit is deleted, then the decoder continues to misplace the codeword boundaries until it receives the codeword 111. However, this behaviour is not catastrophic, for while the decoder is out of synchronisation, it is still decoding correct symbols and another symbol error occurs only when the decoder goes back in synchronisation. Therefore, for a finite number of errors we have a finite number of symbol errors.

Source Symbol	Codeword
a	00
b	111

Table 3.3: Simple VLEC code C_6

In practice, however, catastrophic behaviour is not as serious as it may appear, since this catastrophic behaviour is present only for particular messages. Hence, even if a code is catastrophic, the probability that it will produce an infinite number of symbol errors from a finite number of channel errors is very small. However, catastrophic codes may have longer error propagations.

3.5. Performance

In general, the performance of error-correcting codes is difficult to determine analytically. For this reason two alternative approaches are usually followed. The performance of error-correcting codes may be bounded using the code characteristics (such as the minimum distance in the case of block codes and the free distance in the case of convolutional codes). Alternatively, computer simulations of the particular code on the given channel may be performed. Computer simulations tend to be inadequate for high signal-to-noise ratios, due to the large number of sample points required to obtain an accurate result. On the other hand, bounds tend to be tight at high signal-to-noise ratios but not so tight at the low end. Hence, both these methods are useful to obtain the performance characteristics of codes. In addition, bounds usually give us indications regarding which code parameters influence most the performance of the code. This will then enable us to design good codes.

VLEC codes are no exception to these rules, as we shall see in this section. We shall first derive an upper bound on the symbol error probability of a VLEC code in terms of its free distance.

3.5.1. Union Bounds

A *first error event* is said to occur at an arbitrary bit position r corresponding to state S_r, if the correct path is eliminated *for the first time* at bit position r in favour of a

competitor (the *incorrect path*). Denote the *first error event probability at bit position r* as $P_f(E, r)$. The incorrect path must be some path that had previously diverged from the correct one at some bit position q, $q < r$, and is now converging back for the first time at bit position r.

Let $p^i_{q,r}$ be the correct path segment through the trellis with initial and final states S_q and S_r respectively, with $\| p^i_{q,r} \| = \eta_i$. By definition, $| p^i_{q,r} | = r - q$ and let $N = r - q$. Denote by $\mathbf{f}_i \in F_N$, the codeword of the extended code F_N of order N corresponding to $p^i_{q,r} = (\mathbf{c}_{i_1}, \mathbf{c}_{i_2}, \cdots, \mathbf{c}_{i_{\eta_i}})$, i.e. $\mathbf{f}_i = \mathbf{c}_{i_1}\mathbf{c}_{i_2}\cdots\mathbf{c}_{i_{\eta_i}}$. Note that, as before, we can also say that $p^i_{q,r} = (a_{i_1}, a_{i_2}, \cdots, a_{i_{\eta_i}})$ where, $\mathbf{c}_i$ is the codeword mapped to the source symbol a_i. Further, let $p^j_{q,r}$ be a second path segment through the trellis, with $\| p^j_{q,r} \| = \eta_j$ and with the same initial and final states as $p^i_{q,r}$ and let this be encoded as $\mathbf{f}_j$. Note that $\mathbf{f}_j \in F_N$ as well, since $| p^i_{q,r} | = | p^j_{q,r} | = N$. In general, however, $\eta_i \neq \eta_j$.

Assuming a memory-less source, then the probability of occurrence of $p^i_{q,r}$, $P(p^i_{q,r})$, is given by

$$P(p^i_{q,r}) = P(\mathbf{f}_i) = P(\mathbf{c}_{i_1})\, P(\mathbf{c}_{i_2}) \cdots P(\mathbf{c}_{i_{\eta_i}}) = P(a_{i_1})\, P(a_{i_2}) \cdots P(a_{i_{\eta_i}}). \tag{3.23}$$

Let $h = H(\mathbf{f}_i, \mathbf{f}_j)$. We will also denote this distance by $H(p^i_{q,r}, p^j_{q,r})$. Then, the probability that the path $p^i_{q,r}$ is decoded incorrectly as $p^j_{q,r}$ over the BSC with cross-over probability p, P_h, is given by

$$P_h = \begin{cases} \displaystyle\sum_{e=\frac{h+1}{2}}^{h} \binom{h}{e} p^e (1-p)^{h-e} & h \text{ odd} \\ \displaystyle\frac{1}{2}\binom{h}{\frac{h}{2}} p^{\frac{h}{2}} (1-p)^{\frac{h}{2}} + \sum_{e=\frac{h}{2}+1}^{h} \binom{h}{e} p^e (1-p)^{h-e} & h \text{ even.} \end{cases} \tag{3.24}$$

Lemma 3.2: The first error event probability at any bit position, $P_f(E)$, is bounded by

$$P_f(E) < \sum_{h=d_{\text{free}}}^{\infty} A_h P_h$$

where, A_h is the average number of converging pairs of paths at Hamming distance h and d_{free} is the free distance of the code.

Proof: $P_f(E, r)$ can be bounded, using a union bound, by the sum of the error probabilities of all paths which have the same initial and final states S_q and S_r respectively,

$\forall\ q = 0, 1, \cdots, r-1$ [5]. Now, the joint probability that the correct path is $p^i_{q,r}$ and that this is erroneously decoded as $p^j_{q,r}$ is given by $P_h P(p^i_{q,r})$. Hence,

$$P_f(E,r) < \sum_{q=0}^{r-1} \sum_{(i,j)\in G_{q,r}} P_h P(p^i_{q,r}) \tag{3.25}$$

where, P_h is given by equation (3.24) with $h = H(p^i_{q,r},\ p^j_{q,r})$, $G_{q,r}$ is the set of all pairs of path indices (i, j) such that $i \neq j$ and $\left|\left(p^i_{q,r}\right)_{\beta_i}\right| \neq \left|\left(p^j_{q,r}\right)_{\beta_j}\right|$ $\forall\ \beta_i = 1, 2, \cdots, \eta_i - 1$ and $\beta_j = 1, 2, \cdots, \eta_j - 1$. $\left(p^i_{q,r}\right)_{\beta_i}$ denotes that part of path segment $p^i_{q,r}$ containing the initial β_i transitions, i.e. $\left(p^i_{q,r}\right)_{\beta_i} = (\mathbf{c}_{i_1}, \mathbf{c}_{i_2}, \cdots, \mathbf{c}_{i_{\beta_i}})$. In other words, $G_{q,r}$ is the set of all pairs of path segment indices corresponding to path segments which diverge at state S_q and merge again for the first time at state S_r.

Furthermore, in Section 3.2.2.1 we have seen that all states in the trellis are identical. Hence, we may further overbound (3.25) by considering all paths which merge (at any bit position), i.e.

$$P_f(E,r) < \sum_{k=1}^{\infty} \sum_{(i,j)\in G_{0,k}} P_h P(p^i_{0,k}). \tag{3.26}$$

Note that in this case $h = H(p^i_{0,k},\ p^j_{0,k})$.

Since the RHS of (3.26) is now independent of r, then the first error event at any bit position, $P_f(E)$ is bounded by

$$P_f(E) < \sum_{k=1}^{\infty} \sum_{(i,j)\in G_{0,k}} P_h P(p^i_{0,k}). \tag{3.27}$$

Let A_h be the average number of converging pairs of paths at Hamming distance h, i.e.

$$A_h = \sum_{r=1}^{\infty} \sum_{\substack{(i,j)\in G_{0,r} \\ H(p^i_{0,r},p^j_{0,r})=h}} P(p^i_{0,r}). \tag{3.28}$$

Then, we can re-write (3.27) as

$$P_f(E) < \sum_{h=d_{\text{free}}}^{\infty} A_h P_h \tag{3.29}$$

where d_{free} is the free distance of the VLEC code C as given in Definition 3.11. ■

[5] In general, not all q's are possible, and hence for some values of q, $P(p^i_{q,r}) = 0$.

The set A_h for all possible values of h is said to constitute the distance spectrum for the code.

One may immediately recognise that the bound given by (3.29) is in the same form as that for convolutional codes [Lin & Costello, 1983].

Theorem 3.5: The error event probability, $P(E)$ at any bit position is bounded by

$$P(E) < \sum_{h=d_{\text{free}}}^{\infty} A_h P_h .$$

Proof: The final decoded path can diverge from, and converge with, the correct path any number of times. Figure 3.8 shows the three possible interactions between two incorrect paths p_r^j and $p_r^{j'}$ and the correct path p_r^i. The incorrect path p_r^j contains the incorrect path segment $p_{q,r}^j$, while the incorrect path $p_r^{j'}$ contains the incorrect path segment $p_{q',r'}^{j'}$. When $q > r'$, as in case (a), the two error events are separate. In this case, the first error event occurs at bit position r'. This implies that at state $S_{r'}$, the partial metric for the path segment $p_{q',r'}^{j'}$ is greater than that for the correct path segment $p_{q',r'}^{i}$. A second error event occurs at bit position r. Again, this implies that the partial metric for $p_{q',r}^{j}$ [6] at state S_r is greater than that for $p_{q',r}^{j'}$. This in turn implies that the partial metric for p_r^j is greater than that for the correct path at state S_r. Hence, if p_r^j were compared to p_r^i at bit position r, a first error event would be made. Hence the error event probability at bit position r, $P(E, r)$, is bounded by

$$P(E,r) \le P_f(E,r) . \tag{3.30}$$

The bound given by (3.30) holds also for cases (b) and (c) of Figure 3.8 and hence is a valid bound for any error event occurring at bit position r. In both these cases, the error event at bit position r replaces at least a portion of a previous error event. The net effect may be a decrease in the number of decoding errors (i.e. the number of positions in which the decoded path differs from the correct one). Hence, using the first error event probability as a bound may be conservative in these cases. Substituting for $P_f(E, r)$ in

[6] Note that $p_{q',r}^{j}$ contains also $p_{q',r'}^{j'}$.

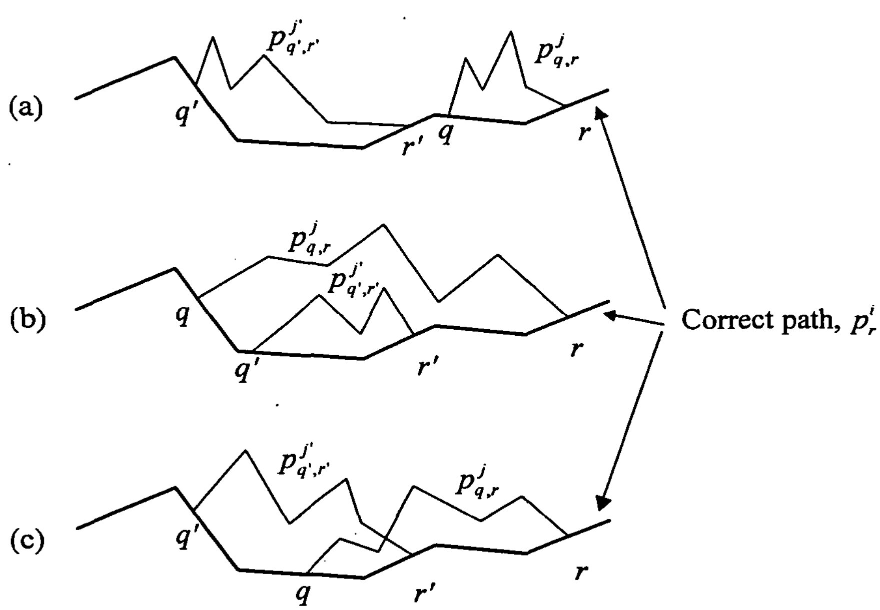

Figure 3.8: The three possible error events interactions

(3.30) and noting that this is independent of r, we have

$$P(E) < \sum_{h=d_{\text{free}}}^{\infty} A_h P_h \,. \tag{3.31}$$

■

In a similar fashion we can obtain a union bound for the symbol error probability of VLEC codes, $P_s(E)$.

Theorem 3.6: The symbol error probability of a VLEC code C is bounded by

$$P_s(E) < \sum_{h=d_{\text{free}}}^{\infty} B_h P_h$$

where, B_h is the average Levenshtein distance of all converging pairs of paths whose encoded messages are at a Hamming distance h from each other.

Proof: Again, let $p^{i}_{q,r}$ be the correct path segment. Let the source symbol sequence associated with $p^{i}_{q,r}$ be $(a_{i_1}, a_{i_2}, \cdots, a_{i_{\eta_i}})$ and let $\mathbf{a}_i = a_{i_1}a_{i_2}\cdots a_{i_{\eta_i}}$. Similarly, let $p^{j}_{q,r}$ be the incorrect path segment and $\mathbf{a}_j = a_{j_1}a_{j_2}\cdots a_{j_{\eta_j}}$, where $p^{j}_{q,r} = (a_{j_1}, a_{j_2}, \cdots, a_{j_{\eta_j}})$. Then, the number of symbol errors in the decoded message is given by $L(\mathbf{a}_i, \mathbf{a}_j)$ (see Section 2.9.1). Hence, averaging the number of symbol errors over all possible pairs of converging paths, we have

$$P_s(E) < \sum_{k=1}^{\infty} \sum_{(i,j)\in G_{0,k}} L(\mathbf{a}_i, \mathbf{a}_j) P_h P(\mathbf{a}_i) \,, \tag{3.32}$$

where, as before, h denotes the Hamming distance between the *encoded* paths $p_{0,r}^{i}$ and $p_{0,r}^{j}$, i.e. $h = H(p_{0,r}^{i}, p_{0,r}^{j})$.

Let B_h be the average Levenshtein distance between all converging path pairs whose encoded messages are at a Hamming distance h from each other, given by

$$B_h = \sum_{k=1}^{\infty} \sum_{\substack{(i,j)\in G_{0,k} \\ H(p_{0,k}^{i}, p_{0,k}^{j})=h}} L(\mathbf{a}_i, \mathbf{a}_j) P(\mathbf{a}_i) \tag{3.33}$$

then,

$$P_s(E) < \sum_{h=d_{free}}^{\infty} B_h P_h \,. \tag{3.34}$$

■

3.5.1.1. Evaluating the Distance Spectrum

In Chapter 6 we shall prove that VLEC codes are always non-linear. In addition, within a set of paths of the same length k, not all path comparisons are allowed, since the pair of paths must have indices which are elements of $G_{0,k}$. Hence, there are no distance invariant VLEC codes. Therefore, to evaluate the distance spectrum for any given code, we must consider all possible paths through the trellis and then take the average as given out in equations (3.28) or (3.33) as required. In the case of convolutional codes, which are linear, this task is much simplified, since we only need to take the all zero path as the correct path, and evaluate the weight spectrum of the other converging paths [Lin & Costello, 1983]. There are several fast algorithms which take advantage of this linearity property of convolutional codes to evaluate their distance spectrum. One such algorithm is the FAST algorithm given by Cedervall and Johannesson [1989]. Rouanne and Costello [1989] give an algorithm to evaluate the distance spectrum of regular and quasi-regular trellis codes, which are non-linear codes. However, to simplify the algorithm they define an equivalence relation which enables them to reduce the problem to a linear one. Unfortunately, this technique cannot be applied in the case of VLEC codes in general. Some slight simplifications may be achieved for some classes of VLEC codes, however these will be not be treated here.

A brief outline of two algorithms developed to calculate the distance spectrum of VLEC codes is now presented. For a more detailed treatment, one should refer to [Buttigieg, 1994a] (see Appendix B).

The first algorithm is optimised for speed, but requires a large memory space. In this algorithm, all possible paths in the trellis starting from state S_0 are extended one after the other, the path p_r^i with r minimum being extended first each time. Each extended path is compared to all the other previously extended paths in the trellis and their intermediate Hamming distances are updated. When two paths converge to the same state, then the Levenshtein distance between the source symbols representing the paths is calculated, together with the probabilities of the two paths. These values are then used as two components[7] in the summations given by equations (3.28) and (3.33). Once two paths converge, then all extended paths from these two are no longer compared with each other. The algorithm is stopped when some pre-determined state is reached.

In order to reduce the memory requirements of this algorithm and increase the speed of operation, we may limit the value of h for which we calculate A_h and B_h, since, as we shall see in Section 3.5.2, only the values of B_h for small h are significant in the bound given by expression (3.34).

The main disadvantage with this algorithm is that we need to store almost all the different paths through the tree representation of the VLEC codes together with their probabilities and their pairwise intermediate Hamming distances.

In the second algorithm, we eliminate the above disadvantage by extending only two paths at a time. The algorithm is outlined below.

1. Set r, the number of bits in the encoded path, to be equal to the shortest codeword in the VLEC code C, i.e. $r = l_1$.
2. Find the first path in the trellis consisting of exactly r bits, i.e. p_r^1. Call this the *reference* path. If no such paths are found go to step 6.

[7] Since if path p_r^i converges to path p_r^j, then path p_r^j also converges to path p_r^i.

3. Find the next path going to state S_r, p_r^i, which converges with the *reference* path for the first time at state S_r, if any. Call this the *secondary* path and proceed to step 4. Otherwise, if no further paths are found, go to step 5.
4. Calculate the Hamming distance between the encoded *reference* and *secondary* paths and the Levenshtein distance between the source messages representing these paths. Calculate also the probability of the *reference* and the *secondary* paths. These values are then used in equations (3.28) and (3.33) to compute the various A_h and B_h respectively. Repeat step 3.
5. Find the next path after the *reference* path going to state S_r, if any, and call this the new *reference* path. If a new *reference* path is found, go back to step 3. Otherwise, proceed to the next step.
6. Increase r by $\gcd(L_1, L_2, \cdots, L_\sigma)$. Repeat from step 2 until r exceeds the required final state.

The memory requirements of the above algorithm are very small. All we need to store at any one time are two paths. However, with respect to the first algorithm, we are repeating several calculations unnecessarily. Many pairs of paths have the same initial transitions and differ only in later ones. With this second algorithm, we need to calculate the Hamming distance between the common transitions each time. We also waste time to find all the paths going to state S_r in a serial fashion. The same paths are repeatedly searched every time that we change the *reference* path and every time we increase r. In practice however, it is found that for a given limited memory space and time constraint, this second algorithm allows us to compute A_h and B_h for longer paths through the trellis than the first algorithm and hence can be more accurate.

As in the case of the previous algorithm, we may increase the speed by limiting the value of h for which we evaluate A_h and B_h.

3.5.2. Simulation

The second method which is used to determine the performance of VLEC codes is that of computer simulation. In this case, average values for the symbol error probability at

various values of SNR are calculated by encoding, corrupting and then decoding a number of messages of some given length and repeating this for a given number of times dependent on the power of the code being used, the message length used and the state of the channel. The technique used is that of Monte Carlo simulation [Jeruchim et al., 1992]. The following procedure is adopted.

1. A source message, **a**, of length η source symbols is generated using the source symbol probabilities.
2. The source message is encoded using the VLEC code C to give the encoded message **x**.
3. The encoded message **x** is corrupted using the BSC to give the corrupted received message **y**.
4. **y** is decoded using the modified Viterbi decoding algorithm given in Section 3.2.2.2 (using either the maximum likelihood or the MAP metric), to give the decoded message **b**.
5. The symbol error probability of **b** as compared to **a** is calculated using the method given in Section 2.9.2.

The above algorithm is repeated for various values of p to give the performance curve for the code.

3.5.3. Comparing Simulation Results and the Union Bound

By comparing the results obtained through simulation to the union bound on the symbol error probability obtained in Section 3.5.1, we can determine the tightness of the bound, and also the validity of the simulation results. Two such comparisons are presented here.

In all the results presented in this thesis, unless otherwise stated, it is assumed that the BSC is derived from the additive white Gaussian noise (AWGN) channel with coherently detected binary phase shift keying (BPSK) modulation. In this case, the

crossover probability p is given by

$$p = Q\left(\sqrt{\frac{2E_b R}{N_o}}\right) \tag{3.35}$$

where E_b is the energy per bit, N_o is the level of the single-sided power spectral density of white noise, R is the code rate given by $\lceil \log_2 s \rceil / L_{\text{average}}$ and $Q(x)$ is the complementary error function given by [Sklar, 1988][8]

$$Q(x) = \frac{1}{\sqrt{2\pi}} \int_x^{\infty} e^{-u^2/2} du. \tag{3.36}$$

Figure 3.9 shows the performance curves for code C_3 (2@5,3; 6@8,3; 3,2) given in Table 2.3 both with source A_1 (also given in Table 2.3) and with a uniform source. The distance spectrum for this code calculated up to state S_{33} and $h = 10$ is given in Table 3.4. One may observe from this figure that the bound is tight for high SNR, as expected.

As a second example, consider code C_7 (1@3,-; 16@10,3; 3,0) given in Table 3.5 together with its distance spectrum as given in Table 3.6 up to state S_{30} and $h = 10$. In this

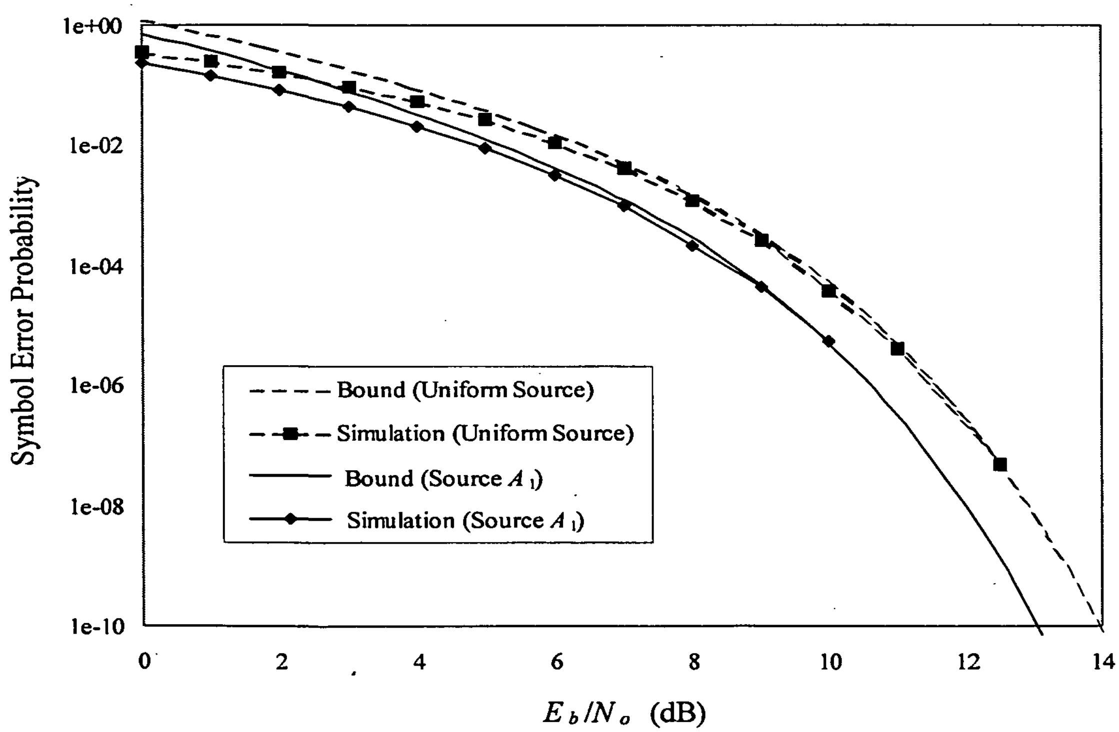

Figure 3.9: Symbol error probability curves for C_3

[8] Note that $\lceil x \rceil$ denotes the smallest integer greater or equal to x.

case, if we compare the bound to the simulation results, we find that the bound is not very tight even at 10dB (see Figure 3.10). The reason for this is that in the bound given by (3.34) we are taking the union of all possible error events, even if some of them overlap. Hence, the symbol error probability is inflated. This becomes more apparent when the number of possible path through the trellis is increased, as in the case of C_7, where the number of codewords is greater than that of C_3. We can, however, tighten the union bound at high SNR as follows.

h	Uniform Source		Source A_1	
	A_h	B_h	A_h	B_h
3	2.50	2.50	1.70	1.70
4	1.50	1.50	0.70	0.70
5	0.07	0.15	0.09	0.18
6	0.70	1.47	0.90	1.84
7	1.62	3.61	1.94	4.18
8	2.72	6.95	3.16	7.55
9	5.19	14.71	5.76	15.18
10	8.84	26.92	9.07	25.93

Table 3.4: Distance spectrum for code C_3 up to state S_{33}

Source Symbols	Codeword
a	000
b	1110000000
c	1110001111
d	1110010011
e	1110011100
f	1110100101
g	1110101010
h	1110110110
i	1110111001
j	1111000110
k	1111001001
l	1111010101
m	1111011010
n	1111100011
o	1111101100
p	1111110000
q	1111111111

Table 3.5: Code C_7

h	Uniform Source	
	A_h	B_h
3	6.60	6.60
4	6.60	6.61
5	0.07	0.15
6	0.29	0.62
7	1.43	2.03
8	0.77	1.89
9	1.48	3.99
10	2.45	6.98

Table 3.6: Distance spectrum for code C_7 up to state S_{30}

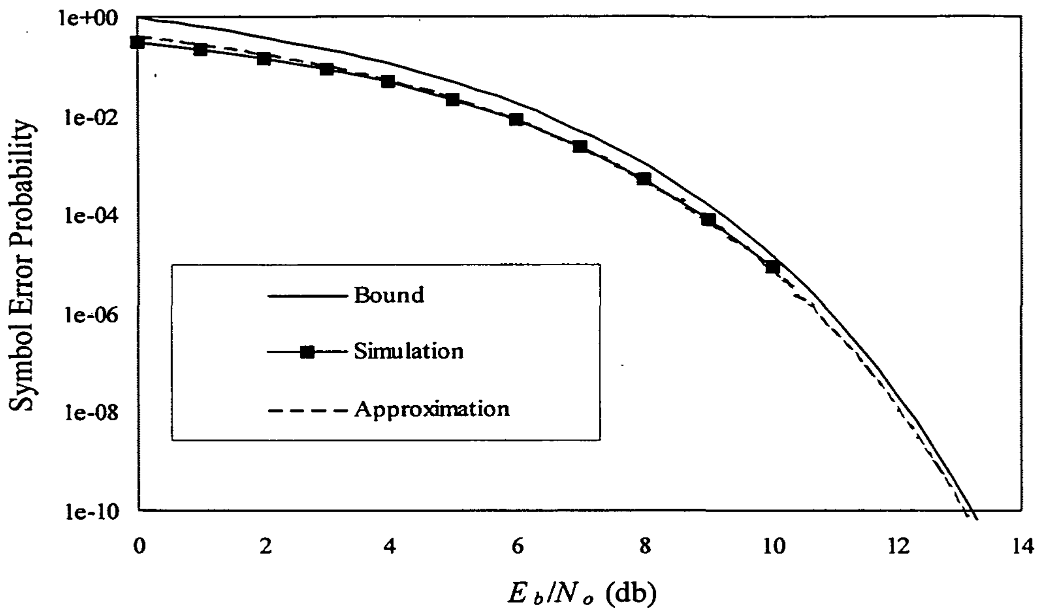

Figure 3.10: Symbol error probability curves for C_7

Corollary 3.1: The symbol error probability of a VLEC code at high SNR is approximately given by

$$P_s(E) \approx B_{d_{\text{free}}} P_{d_{\text{free}}}. \tag{3.37}$$

Proof: For high SNR, p is very small. Hence from equation (3.24) we can deduce that in this case, for even h, $P_h' >> P_{h+1}$. However, for odd h, $P_h = P_{h+1}$. Therefore, for high SNR we can approximate the bound given by (3.34) to

$$P_s(E) < \begin{cases} B_{d_{\text{free}}} P_{d_{\text{free}}}, & d_{\text{free}} \text{ even} \\ \left(B_{d_{\text{free}}} + B_{d_{\text{free}}+1}\right) P_{d_{\text{free}}}, & d_{\text{free}} \text{ odd.} \end{cases} \tag{3.38}$$

Furthermore, since p is very small, then the bit errors are, with high probability, relatively far apart. Hence the error events are non-overlapping. Also, since B_h only includes pairwise comparisons, then a proportion of the error patterns of weight $d_{\text{free}}+1$ are also covered by others of weight d_{free}.[9] Hence, as a first order approximation we can take

$$P_s(E) \approx B_{d_{\text{free}}} P_{d_{\text{free}}} \tag{3.39}$$

for any d_{free} and very small p. ■

The approximate expression for the symbol error probability given by (3.39) for code C_7 is plotted in Figure 3.10. From this figure we can observe that this is indeed a good approximation to the true symbol error probability of the code for high SNR. It is important to note, however, that (3.39) is no longer an upper bound for the symbol error probability at low SNR.

3.5.4. Comparing Maximum Likelihood and MAP Decoding

Theorem 3.7: Maximum likelihood and MAP decoding for a VLEC code C are equivalent iff $l_i/\log_2 P(\mathbf{c}_i) = -\theta \;\; \forall \; \mathbf{c}_i \in C$, where θ is some positive constant, $l_i = |\mathbf{c}_i|$ and $P(\mathbf{c}_i)$ is the probability of occurrence of $\mathbf{c}_i$.

Proof: It may be recalled from Section 3.2.2.2 that the branch metric in the Viterbi decoder for maximum likelihood decoding is $m_{j_{\text{ML}}} = -H(\mathbf{c}_j,\, y_i y_{i+1} \cdots y_{i+l_j-1})$ whereas, from Section 3.3, the branch metric for MAP decoding is $m_{j_{\text{MAP}}}$ = $[\log p - \log(1-p)]H(\mathbf{c}_j,$ $y_i y_{i+1} \cdots y_{i+l_j-1}) + \log P(\mathbf{c}_j)$. Since the branch metrics are applied on codewords of unequal length, then it is simpler to compare the two metrics for equal length paths of, say, N bits. In this case, the two respective metrics at state S_N become, $M_{N_{\text{ML}}} = -H(\mathbf{f}_i, \mathbf{y})$ and $M_{N_{\text{MAP}}}$ = $[\log p - \log(1-p)]H(\mathbf{f}_i, \mathbf{y}) + \log P(\mathbf{f}_i)$ (from equation (3.6)), where $\mathbf{f}_i \in F_N$, F_N being the extended code of C of order N. Hence, for maximum likelihood and MAP decoding of VLEC codes to be equivalent, i.e. for $M_{N_{\text{ML}}}$ to be equal to $M_{N_{\text{MAP}}}$, we require that $P(\mathbf{f}_i)$ is

[9] For example, for the three merged paths (for some code) 00000, 01110 and 11101, the error pattern 01100 in included in B_3 since it gives the decoding error (00000)→(01110). However, it is also included in B_4 since it also gives the decoding error (00000)→(11101). Obviously, in this case only one of these two outcomes is possible, hence the overbound.

constant for all $\mathbf{f}_i \in F_N$. In other words, we require that all codeword sequences of the VLEC code C of total length N bits have the same probability. Now,

$$N = n_1 l_1 + n_2 l_2 + \cdots + n_s l_s \tag{3.40}$$

where, n_i is the number of times the codeword $\mathbf{c}_i$ appears in the N-bit sequence.

Also, for all codeword sequences of total length N bits to be equi-probable, we need that

$$P(\mathbf{c}_1)^{n_1} \times P(\mathbf{c}_2)^{n_2} \times \cdots \times P(\mathbf{c}_s)^{n_s} = 2^{-\frac{N}{\theta}} \tag{3.41}$$

where, θ is some positive constant (it needs to be positive since the LHS of equation (3.41) is always less than 1). Taking logs on both sides of (3.41) and rearranging

$$-n_1\theta \log_2 P(\mathbf{c}_1) - n_2\theta \log_2 P(\mathbf{c}_2) - \cdots - n_s\theta \log_2 P(\mathbf{c}_s) = N. \tag{3.42}$$

Hence, from equations (3.40) and (3.42) we have

$$n_1 l_1 + n_2 l_2 + \cdots + n_s l_s \equiv -n_1\theta \log_2 P(\mathbf{c}_1) - n_2\theta \log_2 P(\mathbf{c}_2) - \cdots - n_s\theta \log_2 P(\mathbf{c}_s) \tag{3.43}$$

Comparing coefficients of $n_1, n_2, \cdots, n_s$ in (3.43), we have that (3.43) is true iff

$$\frac{l_i}{\log_2 P(\mathbf{c}_i)} = -\theta, \quad \forall i = 1,2,\cdots,s \tag{3.44}$$

■

If the lengths and probabilities of the codewords of a code C do not satisfy (3.44), then MAP decoding will always outperform maximum likelihood decoding. However, we observe from the MAP metric given by expression (3.6) that as $p \to 0$, $|h_m \log p| >> |\log P(\mathbf{f}_m)|$ and hence in this case we may neglect $\log P(\mathbf{f}_m)$ in the MAP metric. We can do this only if d_{free} is odd[10], otherwise, if this is even, there will be some received bit sequences which will be equidistant from two codeword sequences, in which case $h_m \log p$ will be equal for both codeword sequences and hence in this case $\log P(\mathbf{f}_m)$ will make a difference. Hence as $p \to 0$, and for odd d_{free}, maximum likelihood decoding is asymptotic to MAP decoding. How fast this occurs depends on the variability of the factor $l_i / \log_2 P(\mathbf{c}_i)$ and the values of B_h for even h.

[10] Only d_{free} is considered here since $p \to 0$.

Definition 3.13: The MAP factor of a VLEC code C is defined to be

$$\phi_{\text{MAP}} = \frac{\sigma_i\left(\dfrac{-l_i}{\log_2 P(\mathbf{c}_i)}\right)}{L_{\text{average}}} \tag{3.45}$$

where, $\sigma(\ )$ denotes the standard deviation, and L_{average} is the average codeword length with the given codeword probabilities.

It is obvious that if $\phi_{\text{MAP}} = 0$, then we have the required condition for maximum likelihood and MAP decoding to be equivalent. It is conjectured that for any given VLEC code, the larger ϕ_{MAP}, the larger the gain with MAP decoding over maximum likelihood at any given p. This is illustrated here for a single code, but has been observed in many other instances. Code C_8 given in Table 3.7 is a (1@4,-; 4@5,3; 3@6,3; 2,1) VLEC code with free distance 3. With source A_2 its MAP factor is 0.006715, whereas with source A_3 its MAP factor is 0.1373. Figure 3.11 shows a comparison between the performance of C_8 with the two sources, decoded with the MAP and the maximum likelihood metrics. Notice that for source A_2 (for which C_8 has a low MAP factor), maximum likelihood and MAP decoding are practically equivalent. However, for source A_3 (for which C_8 has a high MAP factor) MAP decoding achieves some coding gain over maximum likelihood decoding. However, since the free distance for C_8 is odd, then maximum likelihood decoding is asymptotic to MAP decoding for small p, as can be observed in Figure 3.11.

Source Symbol	Source A_2	Source A_3	Codeword
a	0.20	0.35	0111
b	0.13	0.30	00011
c	0.13	0.10	11101
d	0.13	0.10	01000
e	0.13	0.05	10110
f	0.10	0.05	001011
g	0.09	0.03	100010
h	0.09	0.02	110100

Table 3.7: Different source probabilities for code C_8

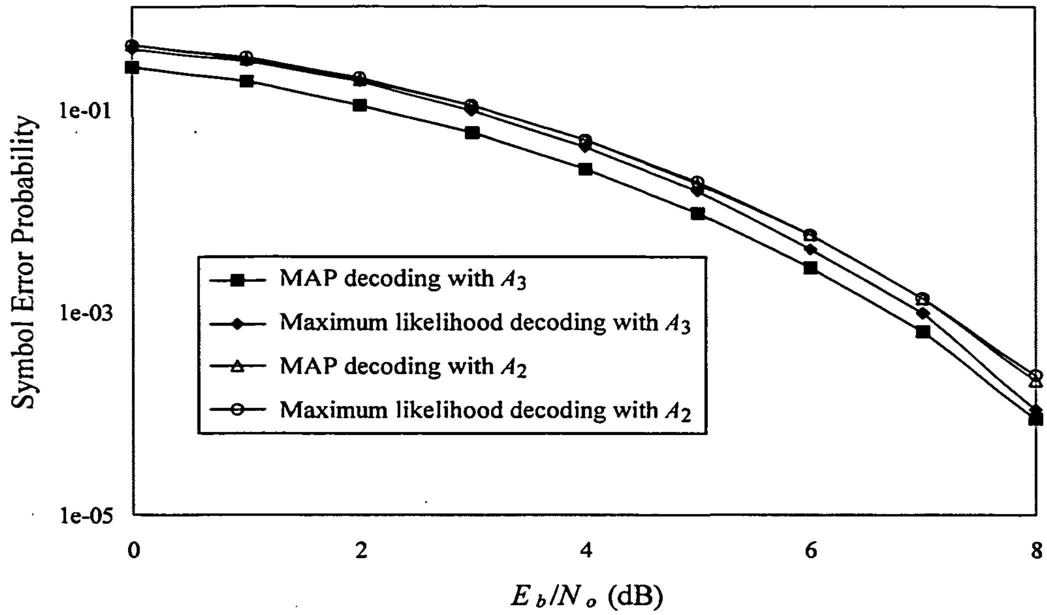

Figure 3.11: Comparisons between MAP and maximum likelihood decoding for C_8

3.5.5. Comparing Maximum Likelihood and Instantaneous Decoding

In Chapter 2 we have given three different instantaneous decoding algorithms for VLEC codes. The prefix decoding and the segment decoding algorithms presented in Sections 2.6.1 and 2.6.3 respectively are appropriate for specific error-mappings. Whereas the tail decoding and the maximum likelihood decoding algorithms, presented in Sections 2.8 and 3.2.2.2 respectively, are more general, even though the former must still be used with instantaneous (or α-prompt) VLEC codes to be effective. In this section we will compare the performance of these decoding algorithms over the BSC.

Table A.1 gives an α_1-prompt code for the 26-symbol English source with a code rate of 0.65. It is also a (13@7,3; 4@10,3; 2@12,4; 2@13,4; 2@14,4; 3@17,3; 3,0) VLEC code with free distance 3. Figure 3.12 shows a performance comparison between the various decoding algorithms for VLEC correcting codes. This code was designed to instantaneously correct a single error per codeword. Notice how prefix decoding outperforms segment decoding. For instance, at a symbol error probability of 10^{-3}, prefix decoding has a coding gain of about 1.5dB. The reason for this is that this code cannot

correct errors in all segments. Also note that prefix decoding and tail decoding (with the original metric derived by Massey[11]) practically have the same performance. However, maximum likelihood decoding achieves a further 1.3dB gain over prefix decoding at a symbol error probability of 10^{-4}. This is achieved at the expense of increased decoding complexity. The main reason why maximum likelihood decoding outperforms the other algorithms is that it is less susceptible to loss of synchronisation. This is not so clear from Figure 3.12, however if we were to observe the error distribution for prefix decoding, say, we will see that there are long periods during which there are no decoding errors. However, when an error pattern outside the error mapping occurs, the decoder decodes a different length codeword, resulting in synchronisation loss and a burst of decoding errors. This effect is especially evident for high SNR. Here, more simulation points are requirèd than is normally expected, since at high SNR the probability of having an error pattern which is not included in the admissibility mapping will become very small.

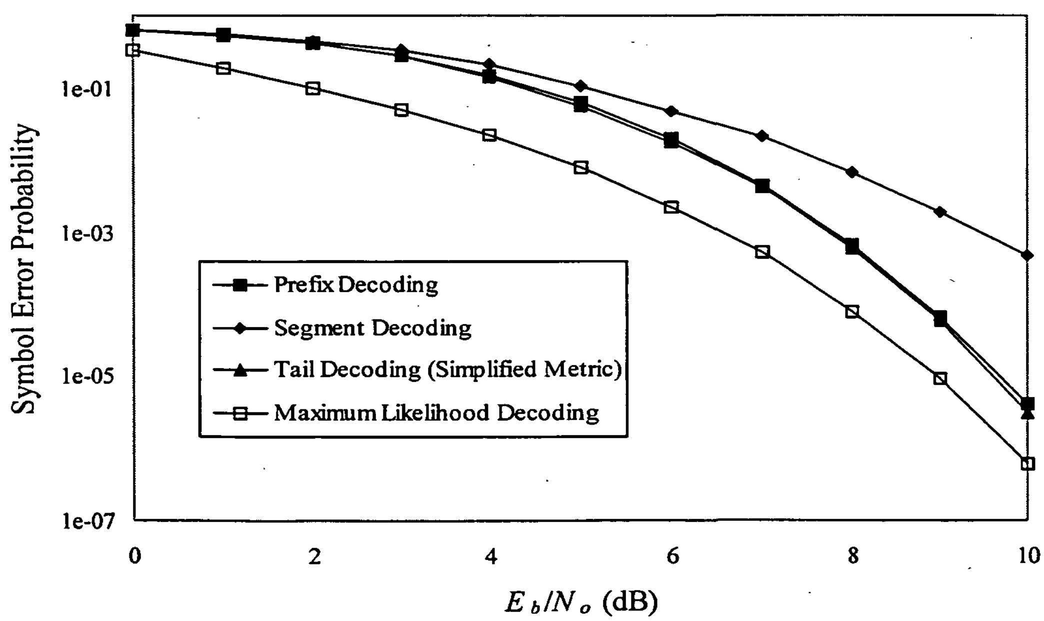

Figure 3.12: Performance comparisons for the α_1-prompt code given in Table A.1

[11] In this case, the bits in the tail are considered to be random bits with probability distribution Q derived from the code and its probability distribution.

Figure 3.13 compares the performance for the various decoding algorithms for the $\alpha_{1,1}$-prompt code also given in Table A.1. This is a rate 0.59 (6@7,3; 20@10,3; 3,0) VLEC code with free distance 3. Note that this time, prefix and segment decoding have the same performance, whereas the tail decoding algorithm has a marginal coding gain over these two algorithms, with the metric derived in Section 2.8 achieving slight gain over the original metric derived by Massey. The real coding gain is achieved by the maximum likelihood decoding algorithm.. For instance at a symbol error probability of 10^{-3} the coding gain is about 1.5dB, over the instantaneous algorithms.

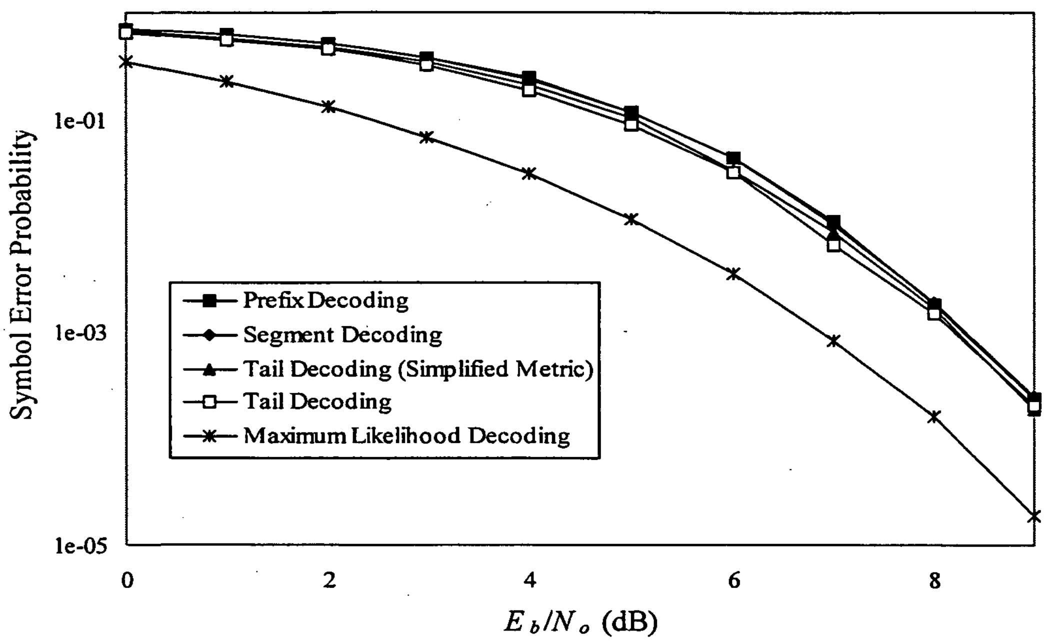

Figure 3.13: Performance comparisons for the $\alpha_{1,1}$-prompt code given in Table A.1

From other results obtained, it was empirically found that prefix decoding always performs better (or the same) as segment decoding. They only have the same performance for codes constructed to correct errors in all segments of the code. Thus we conjecture that prefix decoding is always better than (or equal to) segment decoding. Heuristically, we can explain this by observing that in prefix decoding, the decoding decisions are being based

over a larger number of bits than in segment decoding, which only consider a segment at a time.

Also as expected, the tail decoding algorithm produces better results than both the other instantaneous decoding algorithms. This is also backed by theory, since the tail decoding algorithm is using the optimum metric for instantaneous decoding of VLEC codes. The coding gain achievable by dropping Massey's assumption, that the bits in the tail are chosen independently, is very slight and in practice not worth implementing, since it complicates the metric. However, as expected, the best performance is achieved by the maximum likelihood decoding algorithm.

One further class of VLEC codes that was introduced in Chapter 2 is that of two-length codes introduced by Dunscombe [1988] (see Section 2.7). Here, the difference between the four different decoding algorithms is minimal for most codes considered. For instance, for a two-length code for the 26-symbol English source constructed from the (7,4) Hamming base code by taking 15 codewords from the base code as short codewords and using the 16th codeword of the base code as a prefix to 10 other codewords from the base code to form 16 long codewords of length 14, the performance is practically identical for all four algorithms. The main reason for this is that, as conjectured by Dunscombe, these codes have good synchronisation properties with relatively short error spans. Hence the instantaneous decoding algorithms, in this case, perform almost as well as maximum likelihood. The two main problems with two-length codes are that the code rate cannot be increased much for a given error-correcting capability, since the choice of the base code is limited. The second problem is that these codes are still very similar to fixed-length codes. As we shall see in Chapter 5, these codes are not self-synchronising when the channel admits bit deletions and insertions.

3.6. Decoding Window Depth

In the modified Viterbi algorithm as given in Section 3.2.2.2, the decoder has to wait till the end of the message to start decoding. For large N this will cause a very long decoding delay, which in most cases would be unacceptable. However, as in the case of

convolutional codes, we can limit the decoding delay by forcing the decoder to decode a symbol after some given number of bits have been received. Let W denote this number of bits, called the *decoding window depth.* The decoding algorithm is modified as follows. When W bits are received (assuming that W is less than N), the decoder chooses that state from the last 2^K states in the trellis with the maximum metric, where K is the constraint length of the code. It then retraces the surviving path corresponding to this state and decodes the first symbol in this path. All the surviving paths in the trellis which do not have as their first codeword the decoded codeword, are deleted from the trellis. This ensures that the same number of bits as that transmitted will be decoded. After the first symbol is decoded, another L bits are required before decoding the next one, where L is the number of bits in the decoded codeword. Hence, a decision is always based on the previous W bits held in the decoding window. In this case, the maximum decoding delay[12] is the time corresponding to W bits.

Ideally, we should choose W such that all the surviving paths in the trellis have the same initial transition. This will ensure that whichever one of the surviving paths is chosen, will not affect the value of the decoded symbol. In practice, we can never achieve this goal, however by choosing W large enough we can ensure that this condition is true most of the time.

Figure 3.14 shows the effect of varying the decoding window depth for the VLEC code C_{15} (1@6,-; 2@7,5; 3@8,5; 4@9,5; 5@10,5; 4@11,5; 4@12,5; 3@13,5; 3,2) with free distance 5 for the 26-symbol English source given in Table A.2. The constraint length for this code is 3.7. As a rule of thumb it has been found that if W is taken to be about five times the maximum codeword length in the code, then the performance will be practically the same as for $W = N$. However this factor decreases somewhat when the gcd of the codeword lengths is not equal to one. For code C_{15}, the maximum codeword length is 13 bits. Hence a decoding window depth of about 65 bits should be adequate. This is clearly confirmed by the performance curves given in Figure 3.14.

[12] Delay is measured from the time the codeword is received until it is decoded.

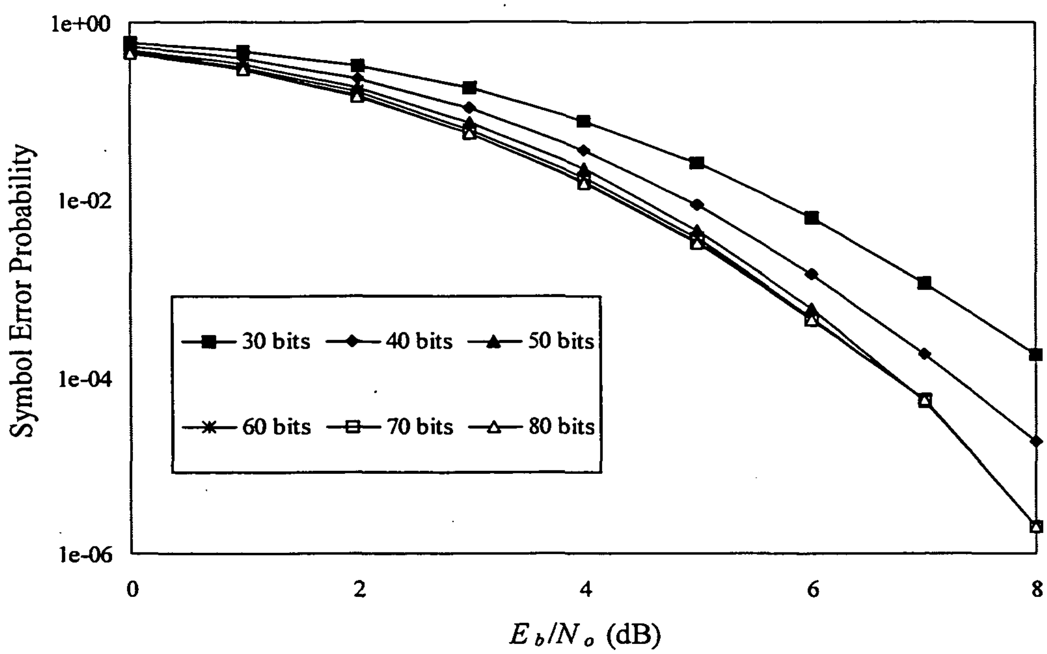

Figure 3.14: Effect of decoding window depth on the performance of VLEC code C_{15}

3.7. Complexity

The question of complexity is always a big issue in any decoding algorithm. To complicate things, not everybody agrees on what should be included and what should be left out in determining the complexity of any particular algorithm. Complexity is important because it determines

- the maximum operating speed of the decoder for a given hardware
- the memory requirements
- the size of the control logic required.

Of the three aspects mentioned above the third one is in general the most difficult to quantify and is greatly influenced by the actual implementation of the algorithm.

Common criteria used to judge the complexity of a coding system based on Viterbi decoding are the numbers of comparisons and additions necessary to decode the transmitted message [McEliece, 1994]. In the case of soft decision decoding, what is important is the number of real operations. Here, we will be dealing with hard decision

decoding, where the operations are all modulo-2. However the results apply equally well for the soft decision case.

In the repetitive part of the trellis for a VLEC code, there are s transitions going to each state. Hence, in the modified Viterbi decoding algorithm as given in Section 3.2.2.2, we need $s-1$ comparisons to determine the one with the maximum metric. Depending on which metric is being used, the comparisons may be either between integers or floating point numbers. However, even in the case of the MAP metric, we can transform the metric to integer values with little degradation in the performance. This is similar to what is done in soft decision Viterbi decoding of convolutional codes where the metrics are also transformed to integers from floating point numbers [Clark & Cain, 1981].

For each transition going into a state, the metric for that transition must be added to the metric of the surviving path from the previous state. This is equivalent to l additions, where l is the number of bits representing the transition. Hence, in the repetitive part of the trellis, we need to perform $\sum_{i=1}^{s} l_i$ additions at each state.

The total number of states in the trellis depends on the message length. If the total number of bits in the encoded message is N and g is the gcd of the codeword lengths of the code, then the total number of states, N_S, for large N, is approximately given by N / g. The approximation arises from the fact that during the initial and final parts of the trellis, some states do not exist. Hence, if the average codeword length for the code C when used to encode some source A is L_{average}, then for large N,

$$N_S \approx \frac{\eta_m L_{\text{average}}}{g} \tag{3.46}$$

where, η_m is the number of source symbols in the message.

Hence, using expression (3.46) we obtain the following relationships for the number of comparisons and additions required to perform maximum likelihood decoding of VLEC codes.

$$\text{Number of comparisons per source symbol} \approx \frac{L_{\text{average}}(s-1)}{g} \tag{3.47}$$

$$\text{Number of additions per source symbol} \approx \frac{L_{\text{average}} \sum_{i=1}^{s} l_i}{g} \tag{3.48}$$

The memory requirements for the modified Viterbi decoding algorithm depends to a large extent on the depth of the decoding window, W. In practice, all we need to store in this case is W/g source symbols. In addition, we need to store the metric of the surviving paths. To this end, we need to store 2^K metric values, where K is the constraint length of the VLEC code.

3.8. Conclusion

In this chapter we have given a maximum likelihood decoding algorithm for VLEC codes which is not, however, instantaneous. We have also derived a MAP metric for VLEC codes. However, in practice, the coding gain achievable by using the MAP metric instead of the simpler Hamming distance metric for most codes and sources is negligible, especially at high SNR. Hence, MAP decoding is not worth implementing due to the increased complexity in computing the path metrics. Note also that MAP decoding of VLEC codes requires the decoder to know the state of the channel[13], something that is not always possible. The increased coding gain achievable by maximum likelihood decoding over the instantaneous decoding algorithms presented in Chapter 2 is, however, significant and this warrants the use of maximum likelihood decoding for VLEC codes over noisy channels. The question of how the performance of VLEC codes with maximum likelihood decoding compares with standard concatenated schemes employing separate source and channel coding will be dealt with in Chapter 6.

We have also seen that the modified Massey metric together with the tail decoding algorithm given in the previous chapter is really optimal only in the instantaneous case. In many cases, the coding gain achievable with this algorithm over the prefix decoding algorithm is minimal. The metric used is more complex, though. Hence, its practical use is minimal, especially in view of the larger gain achievable with maximum likelihood.

[13] This is so because p, the cross-over probability, is part of the MAP metric.

An interesting aspect which comes out from the derivation of the union bound on the symbol error probability of VLEC codes given in Section 3.5.1 is that it agrees perfectly with the way in which the symbol error probability was calculated in Section 2.9.2, where the Levenshtein distance between the transmitted message and the decoded one is approximated by effectively segmenting the received message into separate error events and evaluating the Levenshtein distance separately on these segments. This also seems to us the most natural way in which to calculate the symbol error probability. However, as already commented on in Chapter 2, the difference between this and the original definition for symbol error probability is negligible in practice.

The definition for the free distance of a VLEC code gives us also another way in which to say that a code is uniquely decodable. A code with free distance of one or more implies that all paths in the trellis are distinguishable and hence the code is uniquely decodable. Obviously, for the code also to be able to correct errors, we require that the free distance be at least equal to three.

The constraint length of a VLEC code, as in the case of convolutional codes, is directly related to the number of states required in the decoder for optimal decoding. It also indirectly influences the free distance of the code, since it is directly related to the codeword lengths. In convolutional codes, the constraint length also has a direct bearing on the required decoding window depth in the Viterbi decoder. A similar relation also seems to exist for VLEC codes, however more experimental data is required to get a more exact picture. We have already commented that as a rule of thumb, the decoding window depth for VLEC codes should be about five times the maximum codeword length. However, when the gcd of the codeword lengths is not equal to one, this factor is less. In the extreme case, when we have two-length codes and the constraint length is one, a decoding window depth equal to the maximum codeword length (instantaneous decoding) is sufficient for near maximum likelihood decoding.

Chapter 4.

Sequential Decoding

4.1. Introduction

The maximum likelihood decoding algorithm for VLEC codes given in Chapter 3 searches through all the possible paths in the code tree of a given length and chooses the one at the minimum Hamming distance to the received sequence. It does this in the most efficient way possible by using the modified Viterbi algorithm. This ensures that paths in the tree which cannot form part of the maximum likelihood path are discarded as soon as possible, and hence this limits the search space in the code tree.

Another way to reduce the search space in the code tree is to do a sequential search by extending in turn those paths of the code tree which are most likely. If the distance between any two paths in the code tree increases with increasing path length (by effectively using a non-catastrophic VLEC code), then, if a wrong path is chosen at some point, its distance to the correct path will continue to grow with the path length. Eventually, the decoder notices that this is no longer a good path and backtracks to some previous path. Hence, if there are no errors on the channel, the correct path is extended immediately and the decoding process is much faster than with the modified Viterbi algorithm. However, if the number of errors is large, the decoder may search a large proportion of the code tree before finding the correct path (if this is possible), thus becoming more complex than the modified Viterbi algorithm.

In this chapter we shall derive an optimum metric which determines the most likely path to follow in sequential decoding of VLEC codes. The stack algorithm is adapted to

perform sequential decoding of VLEC codes [Buttigieg & Farrell, 1994c]. We shall also characterise the necessary requirements for near optimal decoding. The computational effort required for sequential decoding will be compared with that required for maximum likelihood decoding and a condition for this to be less, given. Due to the similarity between VLEC and convolutional codes, this work mirrors similar work done for convolutional codes. However, some important differences will be highlighted.

4.2. Metric for Sequential Decoding

In Section 3.2 we have found that for maximum likelihood decoding we must decode that path in the code tree which is at the minimum Hamming distance to the received bit sequence. Hence, it is reasonable to assume that for sequential decoding the required metric must be a function of this distance as well. However, unlike in the case of the modified Viterbi algorithm, here we will be comparing paths with unequal numbers of bits. As a result, even a very good, but long, path may be at a greater Hamming distance to the received sequence than a much shorter, very bad path. Hence, we must also take into account the length of the paths being considered, biasing our decisions in such a way that longer paths are preferred, since these would be much more likely. Therefore, the required metric must also be a function of the path length.

In Section 2.8 we dealt with Massey's optimum metric for variable-length codes [Massey, 1972] which, with further manipulations, reduces to the Fano metric, heuristically first introduced by Fano [1963]. As is well known, this is the preferred metric in sequential decoding of convolutional codes [Lin & Costello, 1983]. Hence, it is reasonable to expect that this metric, slightly modified, may also be useful for sequential decoding of VLEC codes.

Consider a VLEC code C ($s_1@L_1$, b_1; $s_2@L_2$, b_2; $\cdots$; $s_\sigma@L_\sigma$, b_σ; $d_{\min}$, $c_{\min}$). Expanding the context of equation (2.20) given on page 55 to include a sequence of codewords, and assuming that the message sent has N bits, we have

$$\Pr(\mathbf{u}_m, \mathbf{y}) = P_m \prod_{i=1}^{N_m} P(y_i | u_{m_i}) \prod_{j=1}^{N-N_m} P_0(y_{N_m+j}) \tag{4.1}$$

where $P_0(y_i)$ is given by equation (2.21) on page 55, $\mathbf{u}_m$ is a path through the tree consisting of a sequence of η_m codewords of length N_m bits given by $\mathbf{u}_m = \mathbf{c}_{m_1}\mathbf{c}_{m_2}\cdots\mathbf{c}_{m_{\eta_m}} = u_{m_1}u_{m_2}\cdots u_{m_{N_m}}$, with $\mathbf{c}_{m_1}, \mathbf{c}_{m_2}, \cdots, \mathbf{c}_{m_{\eta_m}} \in C$, $\mathbf{y} = y_1 y_2 \cdots y_N$ is the received N-bit sequence and $P_m = P(\mathbf{c}_{m_1})P(\mathbf{c}_{m_2})\cdots P(\mathbf{c}_{m_{\eta_m}})$ is the probability of path $\mathbf{u}_m$.

Then, following Massey [1972], dividing (4.1) by $\prod_{i=1}^{N} P_0(y_i)$, which is constant for all messages $\mathbf{u}_m$, and taking logs, we have

$$F(\mathbf{u}_m, \mathbf{y}) = \sum_{i=1}^{N_m} \left[\log \frac{P(y_i | u_{m_i})}{P_0(y_i)} + \frac{1}{N_m} \log P_m \right] \tag{4.2}$$

Hence, given a list of unequal length paths, the most probable path is the one which maximises equation (4.2). Therefore, $F(\mathbf{u}_m, \mathbf{y})$ is the required metric for sequential decoding of VLEC codes and is essentially the same as the Fano metric.

For binary VLEC codes over the BSC with cross over probability p, $F(\mathbf{u}_m, \mathbf{y})$ may be written as

$$F(\mathbf{u}_m, \mathbf{y}) = \sum_{i=1}^{\eta_m} \begin{cases} H(\mathbf{c}_{m_i}, \mathbf{y}_{m_i}) \log p + \left[l_{m_i} - H(\mathbf{c}_{m_i}, \mathbf{y}_{m_i}) \right] \log(1-p) + \log P(\mathbf{c}_{m_i}) \\ -W(\mathbf{c}_{m_i}) \log\left[pQ(0) + (1-p)Q(1) \right] \\ -\left[l_{m_i} - W(\mathbf{c}_{m_i}) \right] \log\left[(1-p)Q(0) + pQ(1) \right] \end{cases} \tag{4.3}$$

where $\mathbf{y}_{m_1}\mathbf{y}_{m_2}\cdots\mathbf{y}_{m_{\eta_m}}\mathbf{y}' = \mathbf{y}$, is the decomposition of the received sequence $\mathbf{y}$ with respect to the codeword sequence $\mathbf{c}_{m_1}\mathbf{c}_{m_2}\cdots\mathbf{c}_{m_{\eta_m}}$ with $|\mathbf{y}_{m_i}| = |\mathbf{c}_{m_i}| = l_{m_i}$ for all $i = 1, 2, \ldots, \eta_m$, $W(\mathbf{c}_{m_i})$ is the Hamming weight of codeword $\mathbf{c}_{m_i}$ and $Q(0)$, $Q(1)$ are the probabilities of sending a 0 and a 1 respectively. For an efficient code, $Q(0) \approx Q(1)$. If we take this approximation, the metric given by equation (4.3) is simplified to

$$F(\mathbf{u}_m, \mathbf{y}) = \sum_{i=1}^{\eta_m} \left\{ H(\mathbf{c}_{m_i}, \mathbf{y}_{m_i}) \log p + \left[l_{m_i} - H(\mathbf{c}_{m_i}, \mathbf{y}_{m_i}) \right] \log(1-p) + \log P(\mathbf{c}_{m_i}) - l_{m_i} \log \tfrac{1}{2} \right\} \tag{4.4}$$

4.3. Stack Algorithm

The modified Fano metric given by equation (4.3) (or the simplified version given by (4.4)) is the log probability of a path $\mathbf{u}_m$ in the code tree of a VLEC code, given the received sequence $\mathbf{y}$. Hence, given a set of paths through the code tree, we can determine the most likely path transmitted, by choosing that one with the maximum metric. Having

decided which of the current paths is the most likely, we then extend this path by considering the next branches (codewords) on this path. This will give a new set of paths searched in the code tree and the process is repeated until a path of length N bits is found. This is the idea behind the stack algorithm used for the sequential decoding of convolutional codes introduced independently by Jelinek [1969] and Zigangirov [1966]. In this algorithm the searched paths and their metrics are stored sorted in a stack, hence the name. Here, this is simply extended for a VLEC code C ($s_1@L_1$, b_1; $s_2@L_2$, b_2; $\cdots$; $s_\sigma@L_\sigma$, b_σ; $d_{\min}$, $c_{\min}$).

1. Put the path representing bit position 0 (the root node of the code tree) in the stack and assign a metric 0 to this path.
2. Extend the path at the top of the stack thus generating s new paths and compute the metrics of these new paths using equation (4.3) (or (4.4), as the case may be). This is simply the metric of the path at the top of the stack added to the metric for each respective codeword.
3. From the paths generated in step 2 retain only the maximum metric paths for each different codeword length. Hence, the number of extended paths is reduced from s to σ.
4. Delete the top path from the stack.
5. Insert the paths retained in step 3 in the stack in such a way that the stack contains paths with decreasing metric values.
6. If the top path in the stack has the same number of bits as that transmitted, then stop and output the information symbols corresponding to this path. Otherwise, repeat from step 2.

Since the decoded path may not be the correct path, then we may have the situation that the decoded path will not have the same number of bits as that transmitted. Hence, in this case care should be exercised at step 6 in the above algorithm, because it may be that the path at the top of the stack will never achieve this condition. This may be solved in two ways. If the decoded path is allowed to have an unequal number of bits, then the algorithm is stopped when the top path has the same number of bits as that transmitted or more.

However, a neater solution is the following. If we observe the trellis structure of a VLEC code, we note that during the initial and final parts of the trellis there are "forbidden" states[1]. For example, for code C_4 given in Table 3.1, from the initial part of the trellis shown in Figure 3.3, the initial forbidden states are S_1, S_2, and S_5. Similarly, if the total number of bits transmitted is N, then the states S_{N-1}, S_{N-2} and S_{N-5} will also be forbidden, since any path to these states cannot lead to a final path of length N bits. Hence, if all the extended paths to the forbidden states are immediately discarded as soon as they are generated, then the above algorithm will always ensure that a final path of length N bits, equal to that transmitted, will eventually reach the top of the stack. This has the added advantage that the last codewords transmitted are offered the same protection level as the others.

Another important thing to note is that following this algorithm, the size of the stack grows at each iteration by $\sigma - 1$ entries. However, as we shall see, in practice it is found that the size of the stack can be limited to a relatively small value without appreciably degrading the performance of the decoder. In this case, those paths with the smaller metric values are simply deleted from the stack.

The following example will clarify some of the points raised above.

Example 4.1: Consider code C_3 given in Table 2.3 and consider that the message *fcbfcdaa*... is transmitted and that the sequence

01**000**010.11001111.10110.01101010.11001111.01111111.00000.00000...[2]

is received over the BSC channel with crossover probability $p = 0.01$. For code C_3 used to encode source A_1 (also given in Table 2.3), the probabilities of transmitting a 0 or a 1 are respectively $Q(0) = 0.5256$ and $Q(1) = 0.4744$. Hence, we can express the metric given by equation (4.3) as

$$F(\mathbf{u}_m, \mathbf{y}) = \sum_{i=1}^{\eta_m} \left\{ -6.629 H(\mathbf{c}_{m_i}, \mathbf{y}_{m_i}) + \log_2 P(\mathbf{c}_{m_i}) + 0.915 l_{m_i} + 0.145 W(\mathbf{c}_{m_i}) \right\}. \qquad (4.5)$$

[1] Here we are not including those states that are always skipped throughout the trellis arising from the fact that the gcd of the lengths of the code would not be one.

[2] As usual, the bits received in error are shown in bold and the codeword boundaries are denoted by '.'.

The related stack evolution is shown in Figure 4.1, where paths present in the stack expressed as a symbol sequence are shown with the corresponding metric values. Notice that up to Step 6, the incorrect path *agbcgc* is being followed. This will in fact form the initial part of the decoded message if the decoder is an instantaneous one as given in Section 2.8 using the Massey metric in its original form as given in equation (2.20).

Step 1	*Step 2*	*Step 3*	*Step 4*	*Step 5*
a -2.37	*ag* -0.41	*agb* -2.63	*agbc* -3.95	*agbcg* -1.99
f -6.91	*f* -6.91	*f* -6.91	*f* -6.91	*f* -6.91
	aa -13.74	*agd* -10.82	*agd* -10.82	*agd* -10.82
		aa -13.74	*aa* -13.74	*aa* -13.74
			agba -14.00	*agba* -14.00
				agbca -15.32

Step 6	*Step 7*	*Step 8*	*Step 9*	*Step 10*
agbcgc -3.11	*f* -6.91	*fc* -3.54	*fcb* -1.27	*fcbf* 1.21
f -6.91	*agbcgcb* -10.03	*agbcgcb* -10.03	*agbcgcb* -10.03	*agbcgcb* -10.03
agd -10.82	*agd* -10.82	*agd* -10.82	*agd* -10.82	*agd* -10.82
agbcgb -13.40	*agbcgch* -10.84	*agbcgch* -10.84	*agbcgch* -10.84	*agbcgch* -10.84
aa -13.74	*agbcgb* -13.40	*agbcgb* -13.40	*agbcgb* -13.40	*fcba* -12.63
agba -14.00	*aa* -13.74	*aa* -13.74	*aa* -13.74	*agbcgb* -13.40
agbca -15.32	*agba* -14.00	*agba* -14.00	*agba* -14.00	*aa* -13.74
	agbca -15.32	*agbca* -15.32	*agbca* -15.32	*agba* -14.00
		fa -18.28	*fa* -18.28	*agbca* -15.32
			fcd -18.65	*fa* -18.28
				fcd -18.65

Step 11	*Step 12*	*Step 13*	*Step 14*
fcbfc 4.58	*fcbfcd* 8.06	*fcbfcda* 10.18	*fcbfcdaa* 12.29
agbcgcb -10.03	*fcbfcb* -6.84	*fcbfcb* -6.84	*fcbfcb* -6.84
fcbfa -10.15	*agbcgcb* -10.03	*agbcgcb* -10.03	*agbcgcb* -10.03
agd -10.82	*fcbfa* -10.15	*fcbfa* -10.15	*fcbfa* -10.15
agbcgch -10.84	*agd* -10.82	*agd* -10.82	*agd* -10.82
fcba -12.63	*agbcgch* -10.84	*agbcgch* -10.84	*agbcgch* -10.84
agbcgb -13.40	*fcba* -12.63	*fcbfcdc* -11.82	*fcbfcdc* -11.82
aa -13.74	*agbcgb* -13.40	*fcba* -12.63	*fcba* -12.63
agba -14.00	*aa* -13.74	*agbcgb* -13.40	*agbcgb* -13.40
agbca -15.32	*agba* -14.00	*aa* -13.74	*aa* -13.74
fa -18.28	*agbca* -15.32	*agba* -14.00	*agba* -14.00
fcd -18.65	*fa* -18.28	*agbca* -15.32	*agbca* -15.32
	fcd -18.65	*fa* -18.28	*fa* -18.28
		fcd -18.65	*fcd* -18.65
			fcbfcda? ????

Figure 4.1: Evolution of stack contents in example 4.1

Clearly in this case, the symbol error probability will be large, because the first codeword is decoded to a different length codeword causing loss of synchronisation. However, this loss of synchronisation is indirectly detected by the stack algorithm in Step 7, when the metric of the incorrect path *agbcgcb* falls below that of the correct path *f*.

Notice also from Figure 4.1 that when there are no errors on the channel, the metric of the correct path increases, whereas the metric of the incorrect paths generally decreases (but see Section 4.5). Note that in Step 14 we require more bits to determine the path *fcbfcda?*.

4.4. Performance

The algorithm presented in the previous section, although it maximises the probability of choosing the most likely path to extend at each state, does not necessarily choose the most likely path to the final state in the trellis, as is the case of the modified Viterbi decoder given in Chapter 3. This notwithstanding any physical constraints imposed on the algorithm so as to ensure limited computation time and buffer space. Simulation results have shown, however, that if the VLEC code is properly designed, then the two algorithms will practically have the same performance.

For the case of maximum likelihood decoding, we have seen in Section 3.5.1 that the performance of a VLEC code is greatly influenced by the value of the free distance of the code. This is also a well-known result for the case of convolutional codes [Viterbi, 1971]. Chevillat and Costello [1978] have also shown that this is also true for sequential decoding of convolutional codes provided that the distance growth of the code exceeds some lower limit. For this reason they define a column distance function for convolutional codes, which measures the minimum Hamming distance between merged and unmerged codewords of the convolutional code of a given length. We shall now define a similar function for VLEC codes.

4.4.1. Column Distance Function

Definition 4.1: The *column distance function* (CDF), $d_c(\eta)$, of a VLEC code is defined as the minimum Hamming distance between any two paths in the code tree with η

symbols (or transitions) in the shorter (in terms of bits) path, with the condition that the first codewords in each path being compared are of unequal length, i.e.

$$d_c(\eta) = \min \{H(p^i_{0,q}, p^j_{0,r}): p^i_{0,q}, p^j_{0,r} \in C^+, q \le r, \| p^i_{0,q} \| = \eta \text{ and } \left|(p^i_{0,q})_1\right| \ne \left|(p^j_{0,r})_1\right| \} \quad (4.6)$$

where, as before, $p^i_{0,q}$ represents a path, or a sequence of codewords, through the tree or trellis starting at state S_0 and ending at state S_q, with $\| p^i_{0,q} \|$ codewords. Since $q \le r$, $H(p^i_{0,q}, p^j_{0,r})$ is the Hamming distance between two possibly unequal length sequences. This is defined to be the Hamming distance between those parts of the sequences of the same length; i.e., $H(p^i_{0,q}, p^j_{0,r})$ is really the diverging distance between the two paths (if the two paths are considered to be codewords).

This definition for the CDF is similar to the one for convolutional codes [Chevillat & Costello, 1978], however due to the variable-length nature of the codes, the codewords being compared may not necessarily be of the same length.

Theorem 4.1: The CDF of a VLEC code is a monotonically increasing function, i.e.

$$d_c(\eta) \le d_c(\eta+1) \quad (4.7)$$

Proof: Remove the last codeword from that path which has η+1 codewords and is at Hamming distance $d_c(\eta+1)$ to some other path in the tree. Now, at best, the minimum Hamming distance of this η-symbol path to all the other (longer) paths in the tree, d_η, remains unchanged or decreased by some value, i.e.

$$d_\eta \le d_c(\eta+1).$$

But, by definition

$$d_c(\eta) \le d_\eta$$

$$\Rightarrow \quad d_c(\eta) \le d_c(\eta+1) \quad (4.7)$$

■

The implication of the CDF is that any path which has η codewords, is at least $d_c(\eta)$ distance away from any other path in the tree, of the same or longer length.

From Theorem 3.3 (page 75), we have seen that the free distance of a VLEC code C $(s_1@L_1, b_1; s_2@L_2, b_2; \cdots; s_\sigma@L_\sigma, b_\sigma; d_{\min}, c_{\min})$ is bounded by

$$d_{\text{free}} \ge \min(b_{\min}, d_{\min} + c_{\min}). \quad (3.18)$$

Now, in the definition for the CDF we have removed from the possible path comparisons, paths whose first codeword are of equal length. This effectively removes *Case* 1 in Theorem 3.3.

Definition 4.2: The ***unequal length free distance***, d_u, of a VLEC code is defined to be the minimum Hamming distance in the set of all arbitrarily long paths that diverge from some common state S_i due to unequal length codewords and converge again in another common state S_j, $j > i$.

Theorem 4.2: The unequal length free distance of a VLEC code $C\ (s_1@L_1, b_1; s_2@L_2, b_2; \cdots; s_\sigma@L_\sigma, b_\sigma; d_{\min}, c_{\min})$ is bounded by

$$d_u \geq d_{\min} + c_{\min}. \tag{4.8}$$

Proof: Since parallel transitions are not considered for the unequal length free distance, then the only case to consider is *Case* 2 of Theorem 3.3. ■

Corollary 4.1: The unequal length free distance of a VLEC code is always greater than or equal to the free distance of the code, i.e.

$$d_u \geq d_{\text{free}} \tag{4.9}$$

Proof: From Theorem 3.3 it is clear that if $b_{\min} \leq d_{\min} + c_{\min}$, then $d_{\text{free}} = b_{\min}$. Hence in this case, since $d_u \geq d_{\min} + c_{\min}$, then $d_u \geq d_{\text{free}}$. Next, if $b_{\min} > d_{\min} + c_{\min}$ then $d_{\text{free}} \geq d_{\min} + c_{\min}$. In other words, the free distance is not influenced by the parallel transitions and hence by Definition 4.2 in this case $d_u = d_{\text{free}}$. ■

Note that for VLEC codes with $\sigma = s$, i.e. all codewords are of different lengths, then $d_u = d_{\text{free}}$.

Theorem 4.3: The CDF of a VLEC code is bounded by

$$d_c(\eta) \leq d_u \tag{4.10}$$

for all η.

Proof: Consider two paths which are at the unequal length free distance d_u.[3] Then, $d_c(\eta') \leq d_u$, where η' is the number of codewords in that path with the least number of codewords, from these two. Now, adding the same codeword to both paths i times, will not increase the distance between the two paths at all. Hence, $d_c(\eta'+i) \leq d_u$ for all $i = 0, 1, 2, \ldots$. Now, since $d_c(\eta)$ is monotonically increasing, then $d_c(\eta) \leq d_u$ for all η. ■

[3] Note that this implies that the first codewords of both paths are of unequal length.

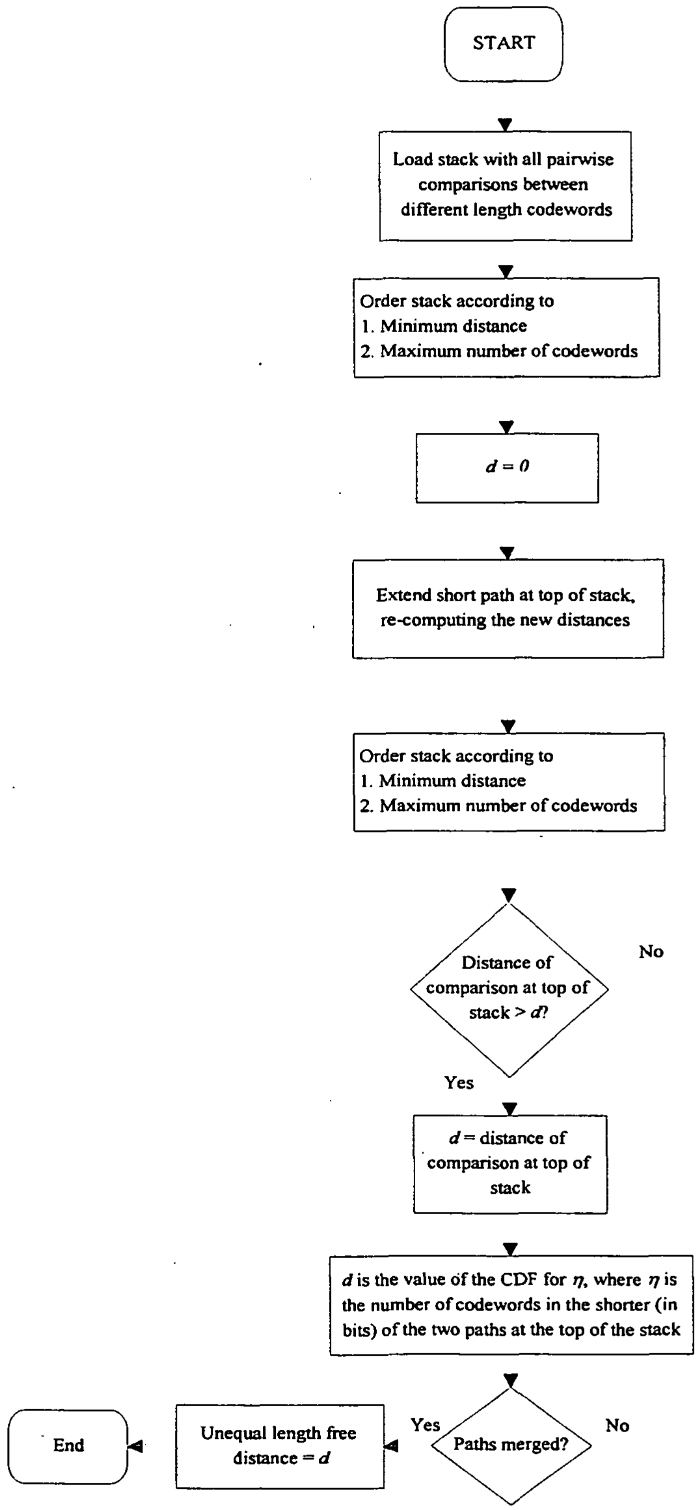

Figure 4.2: Computing the CDF for a VLEC code

Figure 4.2 shows a flowchart for an algorithm to compute the CDF of a VLEC code. This algorithm is derived from a similar one used to compute the CDF of a convolutional code given by Lin and Costello [1983]. It is essentially a sequential search through the code tree using the stack algorithm. The stack contains comparisons between pairs of paths $p_{0,q}^{i}$ and $p_{0,r}^{j}$. Associated with each comparison is the Hamming distance between the two paths $H(p_{0,q}^{i}, p_{0,r}^{j})$ and the number of codewords. This is taken to be the number of codewords in the shorter (in terms of bits) of the two paths. The stack is ordered after each iteration with the comparison having the maximum number of codewords from those having the minimum distance, being placed at the top of stack. The search proceeds by extending each time the shorter of the two paths at the top of the stack. The algorithm ends when the paths at the top of the stack merge, i.e. when they contain the same number of bits. This condition is never reached, however, for the case of sequentially catastrophic codes, defined in Section 4.4.2. In this case, some other criteria must be used to stop the algorithm, such as a maximum number of codewords in the comparison at the top of the stack.

4.4.2. Sequentially Catastrophic VLEC Codes

The CDF does not attain the bound given by expression (4.10) only for certain classes of VLEC codes. For example, for the catastrophic code C_5 given in Table 3.2, $d_u = d_{\text{free}} = 4$ but $d_c(\eta) \le 2$. In fact, it is quite obvious that any catastrophic VLEC code as defined in Section 3.4.3 will have $d_c(\infty) < d_u$. However, there are other codes which exhibit this behaviour. For example code C_6 given in Table 3.3 is not catastrophic as defined in Section 3.4.3, however, for this code $d_c(\eta) \le 2$ for all η, even if $d_u = 4$. As we shall see, this may give problems when sequential decoding VLEC codes.

Definition 4.3: A VLEC code with $d_c(\infty) < d_u$ is defined to be *sequentially catastrophic*.

Suppose we transmit a VLEC code C over the BSC with cross-over probability p. Further, assume that each codeword occurs with equal probability $1/s$, where s is the number of codewords in C. As usual, let L_σ be the length of the maximum length

codeword. Suppose that the metric of the correct path at some state is M_0 and that due to channel errors, the next path to be extended is incorrect due to the fact that the metric M_2 of this path is larger than the metric M_1 of the correct path, as shown in Figure 4.3. Assume that there are no further errors on the channel and that the simplified metric for the stack algorithm given by equation (4.4) is being used. Now, if the metric of the incorrect path remains always greater than M_1, the correct path will never be extended, causing decoding errors[4]. The change in the path metric, ΔM, after each path extension is given by

$$\Delta M = H \log p + (l - H) \log (1 - p) + \log \tfrac{1}{s} - l \log \tfrac{1}{2} \tag{4.11}$$

where l is the length of the codeword used in the path extension and H is the Hamming distance of this codeword to the correct path. Since l depends on the decoded codeword, then it is useful to bound ΔM. For $0 < p < 0.5$,

$$\Delta M \leq H \log p + (L_\sigma - H) \log (1 - p) - \log s - L_\sigma \log \tfrac{1}{2} \tag{4.12}$$

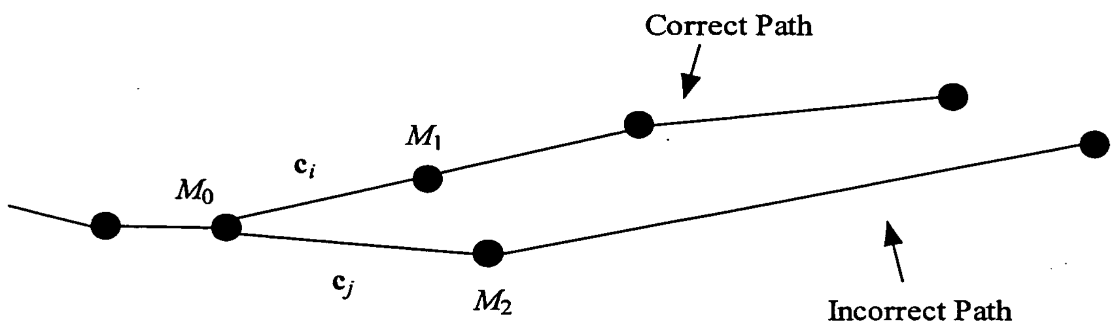

Figure 4.3: Necessary condition for correct decoding with sequential decoding

Hence, if $\Delta M > 0$, then the incorrect path will be extended at the expense of the correct path. Therefore, using the bound given by expression (4.12), incorrect decoding occurs if

$$H \log p + (L_\sigma - H) \log (1 - p) - \log s - L_\sigma \log \tfrac{1}{2} > 0$$

$$\Rightarrow \qquad H < \frac{\log\left\{s\left[2(1-p)\right]^{-L_\sigma}\right\}}{\log\left(\frac{p}{1-p}\right)} \tag{4.13}$$

---

[4] This is not a necessary condition, though, since what is really required is that the metric along the incorrect path is always above the metric of the correct path at some arbitrary position.

Hence, if the CDF for a code does not increase by more than the RHS of expression (4.13), then, for the given cross-over probability, sequential decoding for this code may perform worse than maximum likelihood decoding.

4.4.3. Simulation Results

Code C_9 (1@5,-; 1@8,-; 5, 5) given in Table 4.1 is a sequentially catastrophic code with free distance 10. This is clear from Figure 4.4, which gives the CDF for the code, since the CDF never attains the value of the unequal length free distance (which in this case is the same as the free distance, since $s = \sigma = 2$). Code C_{10} (1@5,-; 1@8,-; 4, 3) is also a free distance 10 VLEC code. However, in this case, the code is non-catastrophic, and any path with five codewords is at least at Hamming distance 10 from all other (longer) paths in the tree. Note that with the same codeword lengths, a free distance 12 code may also be constructed. This is code C_{11} (1@5,-; 1@8,-; 5, 5) also given in Table 4.1. For this code, any path with seven codewords is at least at Hamming distance 12 from all other (longer) paths in the tree.

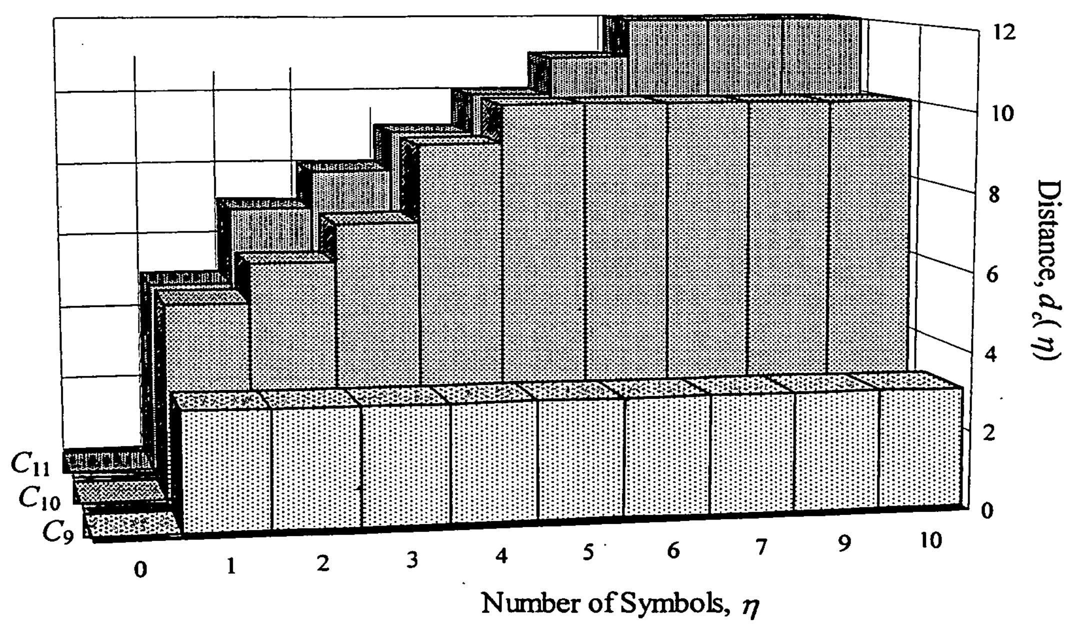

Figure 4.4: Column distance function for the three codes C_9, C_{10} and C_{11}

Source symbol	Code C_9	Code C_{10}	Code C_{11}
a	00000	00110	01101
b	11111111	10001011	10010010

Table 4.1: Two-codeword codes with average codeword length of 6.5 bits for a uniform source

Figure 4.5 shows the performance curves for these three codes with a uniform source, both with maximum likelihood decoding and with sequential decoding using the stack algorithm, with a maximum stack size of 100. Notice that the performance of C_9, which is sequentially catastrophic (but not strictly catastrophic), with sequential decoding is worse than that with maximum likelihood decoding. This difference translates in a 2dB loss in coding gain with sequential decoding at a SEP of 10^{-3}.

With maximum likelihood decoding, however, codes C_9 and C_{10} have practically the same performance, which shows that sequential catastrophicity is not important with maximum likelihood decoding. What is important with maximum likelihood decoding is the free distance, hence the performance of C_{11} is better than that of the other two. This is also true with sequential decoding.

Codes C_{10} and C_{11} are not sequentially catastrophic and for both these codes, the

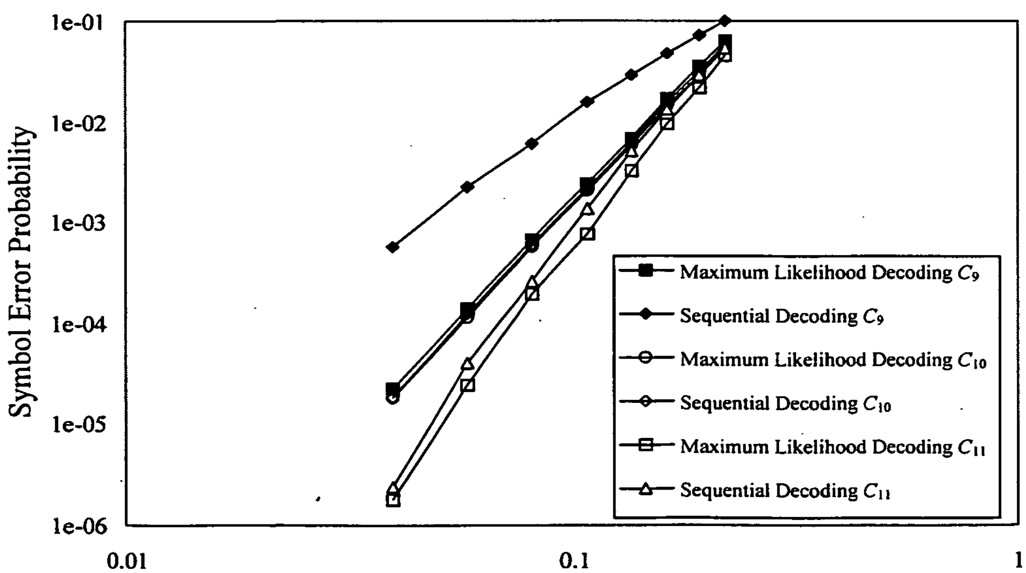

Figure 4.5: Comparing performance of maximum likelihood and sequential decoding for codes C_9, C_{10} and C_{11}

performance curves given in Figure 4.5 show that sequential decoding is asymptotically as good as maximum likelihood decoding. Table 4.2 gives the required CDF growth for these codes, for specific values of p, for the condition given by expression (4.13) to be satisfied. Now, from Figure 4.4, the CDF growth for both C_{10} and C_{11} is approximately one per source symbol. Hence, this requires p to be smaller than 0.01. Note, however, that whereas code's C_{11} performance with sequential decoding is still slightly worse than with maximum likelihood decoding for the range of p shown in Figure 4.5, as expected, that of C_{10} is practically the same. Therefore, care should be exercised when applying the condition given by expression (4.13), due to the simplifying assumptions made in deriving this expression.

Cross-over probability, p	Required CDF growth
0.2000	2.2123
0.1000	1.8246
0.0100	1.0384
0.0010	0.7013
0.0001	0.5267

Table 4.2: Required CDF growth to satisfy condition given by expression (4.12) for codes C_9, C_{10} and C_{11}

As a further example, consider codes C_{15} and C_{17} given in Table A.2. Their respective CDFs are given in Figure 4.6, from which we may deduce that C_{17} has free distance 5 (using also Corollary 4.1) and that C_{15} is sequentially catastrophic. In fact, C_{15} is strictly catastrophic, since the two semi-infinite messages *jazazaz*... and *zhfhfhf*... are at Hamming distance 3 from each other. Figure 4.7 shows the codes' performance both with sequential and maximum likelihood decoding. First of all, it is interesting to note that code C_{15}, which is catastrophic, with maximum likelihood decoding performs almost as well as C_{17}, which is not catastrophic. With sequential decoding, however, C_{15} performs worse than with maximum likelihood whereas the performance of code C_{17} is practically the same with both decoding methods. It is worthwhile noting, though, that the difference in performance for C_{15} is negligible.

Figure 4.8 shows the effect of the stack size for code C_{17} given in Table A.2. Note that for a stack size of just one, the stack algorithm is equivalent to the tail decoding algorithm using the Massey metric as given in Section 2.8. Notice that with a stack size of just fifteen, the performance of the stack algorithm is almost as good as that of maximum

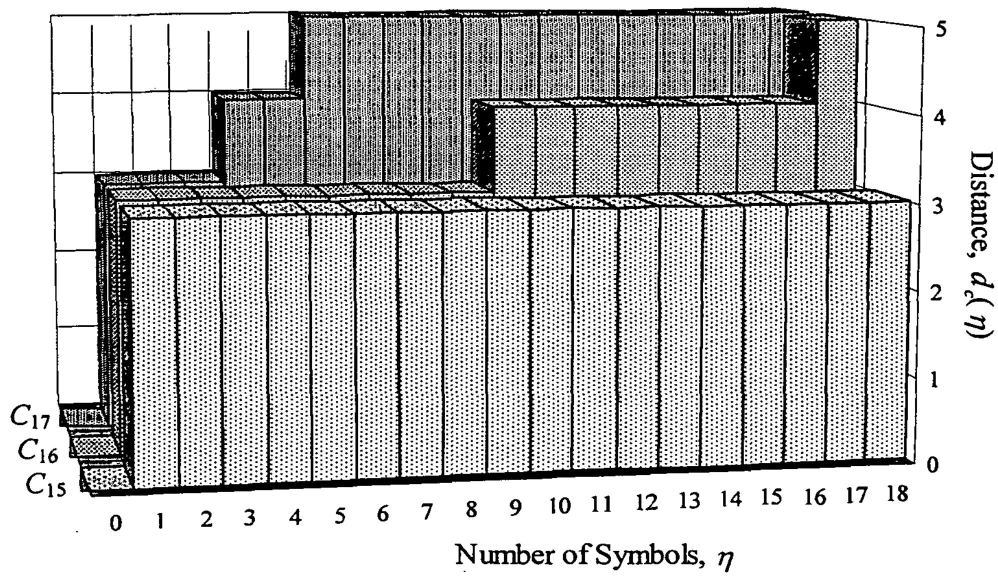

Figure 4.6: Column distance function for the three codes C_{15}, C_{16} and C_{17}

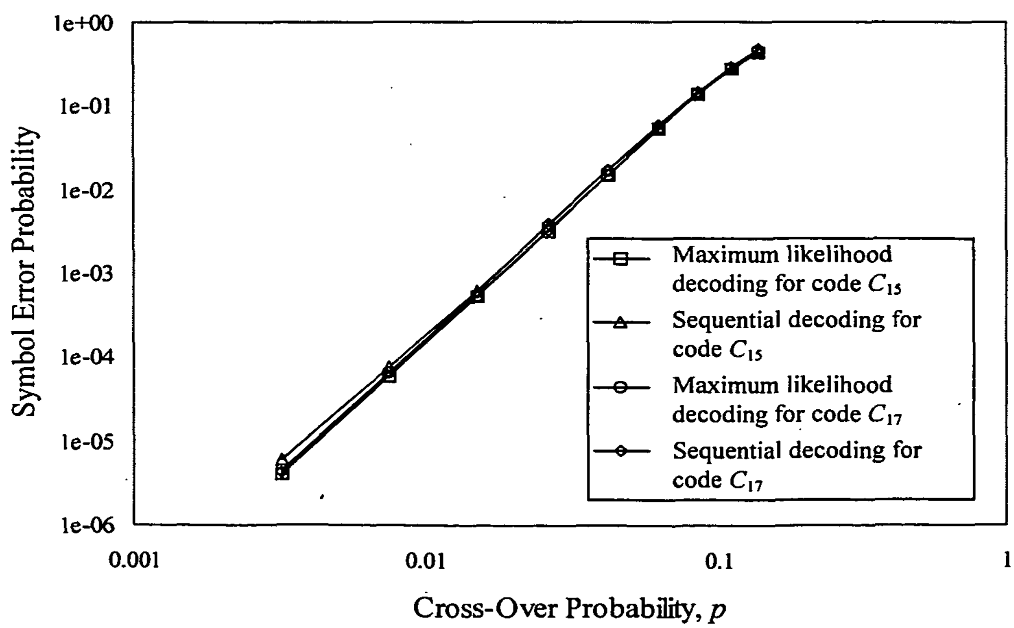

Figure 4.7: Comparison between maximum likelihood and sequential decoding for codes C_{15} and C_{17}

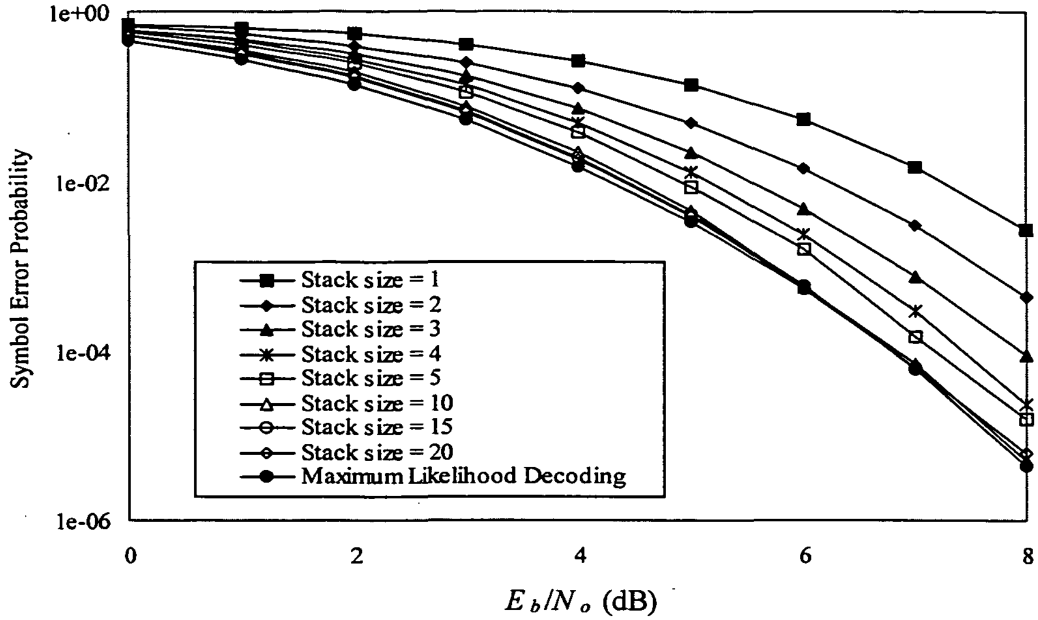

Figure 4.8: Effect of stack size on performance of sequential decoding for code C_{17}

likelihood decoding. In fact, for all non-sequentially catastrophic VLEC codes considered, it has been found that for a relatively small stack size the performance with the stack algorithm is nearly as good as that with maximum likelihood decoding.

Figure 4.9 shows the performance of sequential decoding for code C_{17} with a maximum stack size of 50 using both the full metric as given in equation (4.3) and the approximate metric as given in equation (4.4). Notice that in this case the performance is practically the same with both metrics. The main reason for this is that the probabilities of a '0' and a '1' for code C_{17} when used to encode the 26-symbol English source are nearly equal. In fact, in this case, $Q(0) = 0.5457$ and $Q(1) = 0.4532$. If, on the other hand, we were to consider code C_9 given in Table 4.1 with a uniform source, where $Q(0) = 0.3846$ and $Q(1) = 0.6154$ we would expect that the performance with sequential decoding using the full metric should be better than with the simplified one. In fact this is the case in practice, as shown in Figure 4.10.

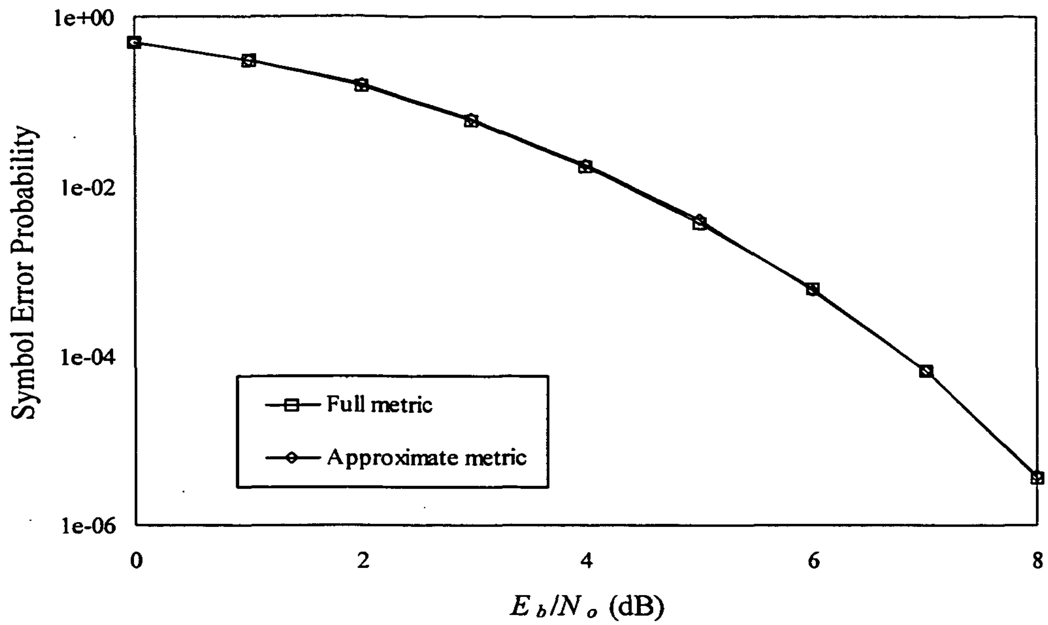

Figure 4.9: Performance of C_{17} with exact and approximate metrics

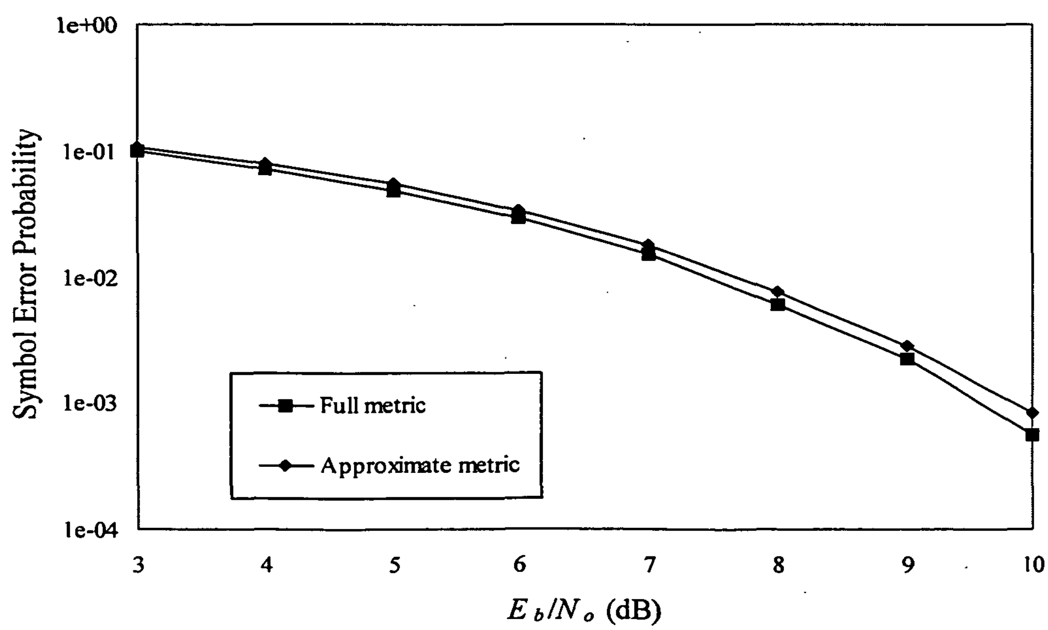

Figure 4.10: Performance of C_9 with exact and approximate metrics

4.5. Complexity

The main objective for developing sequential decoding for VLEC codes is to reduce the decoding complexity. This is, however, not as simple to analyse as in the case of the maximum likelihood decoding algorithm, since the number of paths searched is a random variable dependent on the code used and the state of the channel. In the case of convolutional codes, the computational effort has been upper bounded by Chevillat and Costello [1978]. This has been found to be exponentially dependent on the column distance growth of the code. The more rapidly the column distance grows, the less computational effort required to decode the code. This has also been previously verified experimentally [Chevillat & Costello, 1976]. Other earlier work used random coding arguments to derive an upper bound on the computational effort on the whole ensemble of random tree codes [Savage, 1966a], [Savage, 1966b], [Jacobs & Berlekamp, 1967], [Forney, 1974]. This work has shown that the computational effort is Pareto distributed and that it does not depend on the constraint length of the convolutional code. Although no similar analysis has been attempted for sequential decoding of VLEC codes, it is reasonable to expect that these will exhibit similar properties. However, a few comments are in order.

Figure 4.11 shows the average number of extended paths per transmitted symbol as it varies with the SNR on the channel for codes C_9, C_{10} and C_{11} given in Table 4.1, using a maximum stack size of 100. Ideally, only one path is extended per transmitted symbol. However, in practice this is not possible. The first point to note is that since these codes have two codewords ($s = 2$), then the stack algorithm requires that for each path extended, two new paths are generated[5]. Hence, the minimum number of extended paths for each transmitted symbol, if the correct path always stays at the top of the stack, is two. This occurs when the SNR is high, resulting in few errors on the channel and provided that the metric of the correct path increases at each extension. For the simplified metric given by equation (4.4) this implies that, in the limit when $p \rightarrow 0$, we require that

[5] In general, s new paths are generated.

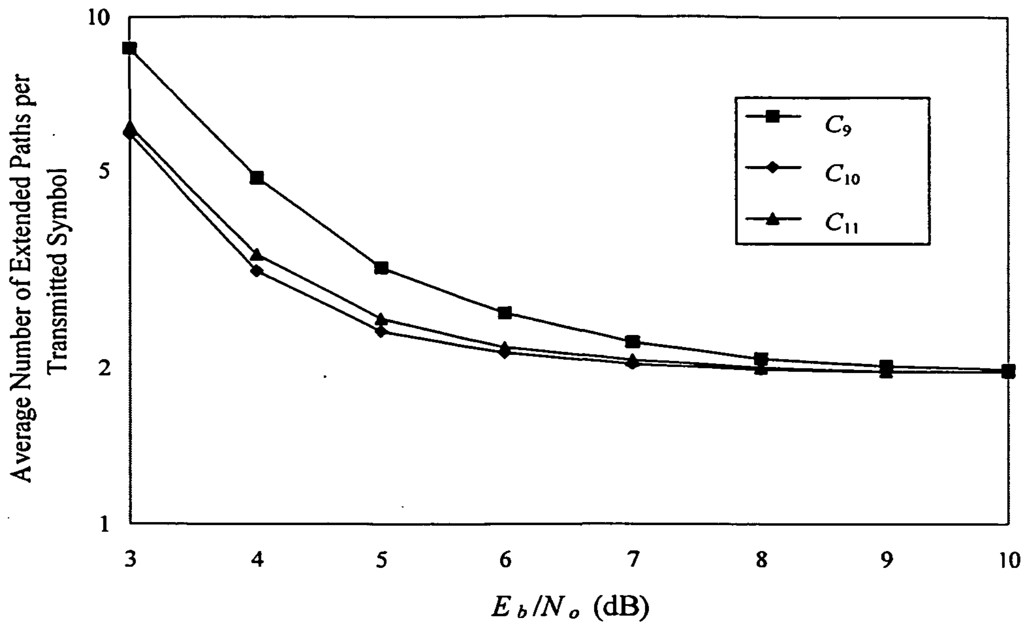

Figure 4.11: Number of extended paths for codes C_9, C_{10}, and C_{11}

$$l_i > \log_2\left[\frac{1}{P(\mathbf{c}_i)}\right] \tag{4.14}$$

For codes C_9, C_{10} and C_{11} with a uniform source, condition (4.14) is satisfied. Hence, the minimum average number of extended paths per transmitted symbol is indeed two, as can be observed from Figure 4.11. Notice that lowering the SNR increases the number of paths that are extended for each transmitted symbol. Figure 4.12 presents a clearer picture. Here, the average number of extra paths which are being extended per source symbol more than the minimum number, in this case two, is plotted against E_b/N_o. Denote this number by E_x. From this figure we may deduce that there is an exponential relationship between the number of extended paths and the value of E_b/N_o; i.e., $E_x \approx a(E_b/N_o)^b$, for some constants a and b.

Assuming that all codewords are considered at each path extension[6], then at each path extension we need to perform $\sum_{i=1}^{s} l_i$ additions to evaluate the new metric for each

[6] This is not exactly true if it is ensured that the decoded message has the same number of bits as that transmitted, since in this case some of the states in the trellis are forbidden and consequently the corresponding transitions are not extended.

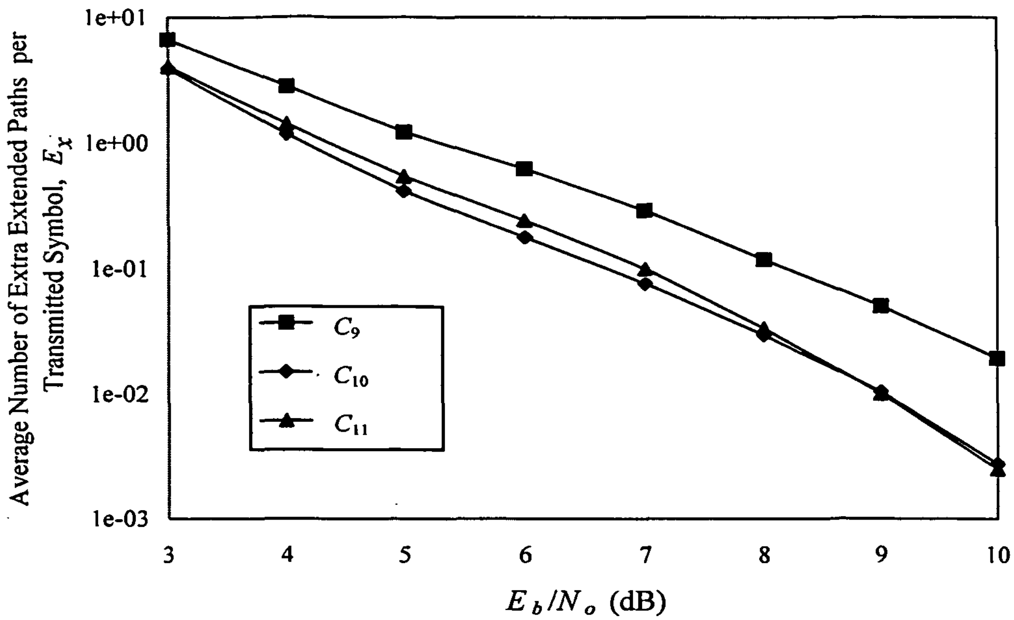

Figure 4.12: Extra extended paths for codes C_9, C_{10} and C_{11}

extended path. Hence, if we let ξ represent the average number of paths which visit the top of the stack per transmitted source symbol, then

$$\text{Average number of additions per source symbol} \approx \xi \sum_{i=1}^{s} l_i \tag{4.15}$$

Note that as $E_b/N_o \to \infty$, $\xi \to 1$ and that

$$\xi = \frac{E_x}{s} + 1 \tag{4.16}$$

In addition, for each path extension, we need to store in the stack that codeword with the best metric for each different codeword length group. Hence, we need to perform $s - \sigma$ comparisons each time that a path is extended. Therefore:

$$\text{Average number of comparisons per source symbol} \approx \xi(s - \sigma) \tag{4.17}$$

Furthermore, each time that a path is extended, σ new paths need to be inserted into the stack in the correct order according to their metric value. However, various techniques may be used to simplify this operation by avoiding actually sorting the entries in the stack. One such technique for the case of convolutional codes is the so-called *stack-bucket*

algorithm [Jelinek, 1969]. For convolutional codes this technique has little effect on the performance of the stack algorithm. Even though this was not tested for VLEC codes, it is expected that the behaviour should be similar. In addition, from the results obtained, some of which presented in Section 4.4.3, it has been found that the size of the stack for the case of VLEC codes need not be large. Hence, the amount of computation in the sorting of the stack can be made relatively small. Therefore, as a first order comparison with the modified Viterbi decoding algorithm for VLEC codes, we shall neglect the computation required in sorting the stack and will just compare the number of additions and comparisons necessary. Hence, comparing equations (4.15) and (4.17) with their respective counterparts for maximum likelihood decoding given by equations (3.48) and (3.47), we may deduce that sequential decoding of a VLEC code is less complex than maximum likelihood decoding if

$$\xi < \frac{L_{\text{average}}}{g} \tag{4.18}$$

where L_{average} is the average codeword length and g is the gcd of the codeword lengths.

For codes C_9, C_{10} and C_{11} with a uniform source, $L_{\text{average}}/g = 6.5$. So, for sequential decoding to be less complex than maximum likelihood decoding, ξ must be less than 6.5. Hence, from the curves shown in Figure 4.12, for E_b/N_o greater than 3dB for codes C_9, C_{10} and C_{11}, sequential decoding is less complex than maximum likelihood decoding with a uniform source. Notice that for code C_9 which is sequentially catastrophic, the average number of extended paths for any given E_b/N_o is larger than for the other two codes. Note also that the number of paths extended for C_{10} and C_{11} are approximately the same and this is in conformity with the fact that the CDF growth for these two codes is approximately equal.

Consider now codes C_{15}, C_{16}, and C_{17} given in Table A.2 used to encode the 26-symbol English source. The respective CDFs are given in Figure 4.6. The variation of E_x with E_b/N_o for these codes is given in Figure 4.13 for a maximum stack size of 50. From this figure it is surprising to note that there is almost no difference between the values of E_x for all three codes, even though these codes have widely differing column distance growth

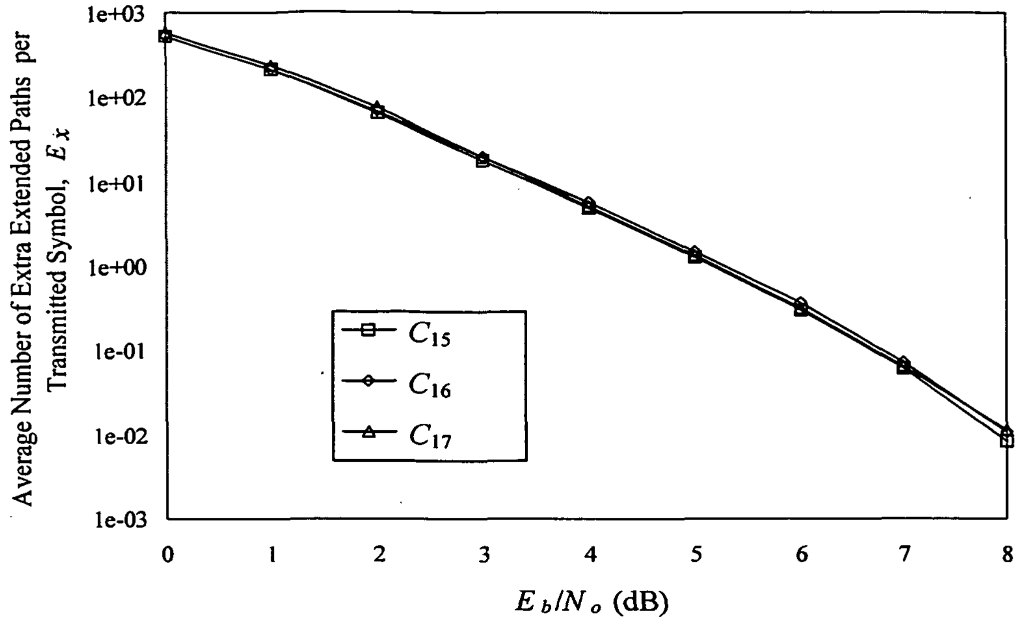

Figure 4.13: Extra extended paths for codes C_{15}, C_{16}, and C_{17}

rates and in fact code C_{15} is even catastrophic. This seems to imply that the CDF growth for the case of VLEC codes is not enough in itself to determine which code is less computationally expensive with sequential decoding.

For these codes, sequential decoding is less complex than maximum likelihood decoding if ξ is greater than about 8.5, or E_x greater than 195. Hence, from Figure 4.13, this occurs when E_b/N_o is greater than 1dB. Table 4.3 gives the computational load for these codes with sequential decoding as a percentage of that required with maximum likelihood decoding, where the computational load is the sum of the total number of additions and comparisons required per source symbol. Notice that for reasonably moderate values of E_b/N_o, the computational load required by the sequential decoding is about a tenth of that required by the modified Viterbi decoding algorithm.

4.6. Conclusion

In this chapter we have given a sequential decoding algorithm, based on the stack algorithm, for VLEC codes. We have seen that the performance of VLEC codes with sequential decoding is as good as that with maximum likelihood decoding, for non-

E_b/N_o	C_{15}/%	C_{16}/%	C_{17}/%
1	106.1	108.2	115.7
2	40.6	41.9	43.8
3	19.5	20.3	19.9
4	13.7	14.0	13.7
5	12.1	12.1	12.0
6	11.6	11.7	11.5
7	11.5	11.5	11.4
8	11.5	11.5	11.4

Table 4.3: Percentage computational load for codes C_{15}, C_{16} and C_{17} with the sequential decoding algorithm as compared with the modified Viterbi algorithm

sequentially catastrophic VLEC codes. However, the complexity for this algorithm can be much less than that required with maximum likelihood decoding, especially for high SNR. Even for E_b/N_o as low as 3dB, equivalent to SEP as high as 0.01, sequential decoding is already less complex than maximum likelihood decoding for the codes considered here.

The metric used in the sequential decoding algorithm is essentially the same as the Fano metric used for convolutional codes. A simplified metric was shown to give comparable results to the exact metric on condition that the channel input symbols occur with approximately equal probabilities. This condition is usually met in practice, since otherwise it would imply that the code is not efficient for the given source.

One interesting aspect found from the experimental results was that the maximum stack size required for near optimal performance is small. A stack size of twenty entries was enough for most codes considered, sometimes even less. As a rule of thumb, the stack size must be chosen to be slightly larger than σ. This is because the free distance paths are, in general (for non-sequentially catastrophic codes), very short. For instance, for code C_{17}, where a maximum stack size of fifteen is enough to give near optimal performance, 97% of the free distance paths contain two codewords or less. This is advantageous since less computational effort will be required to sort the stack.

The column distance function growth, although it does give indications as to whether a code will perform well with sequential decoding, on the other hand does not seem to be a very good test to indicate the computational effort required with sequential decoding, unlike the case with convolutional codes. One problem with the CDF is that it does not

take into account the probabilities of the paths involved. Hence, if the code is catastrophic only for one particular message, then the probability of having catastrophic behaviour is likewise small and thus sequential decoding would perform well in this case. It would be useful to bound the computational effort required in sequential decoding of VLEC codes. This will then indicate which parameters of VLEC codes are most important in this regard.

Another problem with sequential decoding, which was not addressed in this chapter, is the random time required to decode a source symbol. This is a problem with any sequential decoding algorithm, even for convolutional codes. The fact the stack size is small somewhat alleviates this problem.

Chapter 5.

Synchronisation Properties

5.1. Introduction

Loss of synchronisation is said to occur when the decoder does not properly determine the codeword boundaries (see also Section 2.4). In this Chapter we shall be considering four different mechanisms whereby we may have loss of synchronisation in VLEC codes with maximum likelihood decoding. In the first case, considered in Section 5.2, we are going to assume the usual channel model whereby the number of received bits is equal to that transmitted (the BSC). For the other three cases, we will use a different channel model which allows symbol deletions or insertions. The three cases that we will consider are those when: a number of initial bits are lost (Section 5.3); a number of consecutive bits are lost within the message (Section 5.4.1); and a number of consecutive (random) bits are gained within the message (Section 5.4.2).

5.2. Average Error Span on the Binary Symmetric Channel

Consider a VLEC code C ($s_1@L_1$, b_1; $s_2@L_2$, b_2; $\cdots$; $s_\sigma@L_\sigma$, b_σ; $d_{\min}$, $c_{\min}$) transmitted over the BSC with cross-over probability p and decoded using the maximum likelihood decoder of Chapter 3. Using the same notation as introduced in that chapter, consider an error event, whereby the correct path segment $p^i_{q,r}$ associated with the message sub-sequence $\mathbf{a}_i$ is incorrectly decoded into the path segment $p^j_{q,r}$ associated with the message sub-sequence $\mathbf{a}_j$. If $\left\|p^i_{q,r}\right\| = \left\|p^j_{q,r}\right\| = 1$ then the error event causes no loss of synchronisation, since the transmitted codeword is incorrectly decoded into a same length

codeword. However, if $\|p_{q,r}^{i}\| \neq 1$ and/or $\|p_{q,r}^{j}\| \neq 1$ then this would indicate loss of synchronisation. The number of source symbol errors in an error event is simply $\|p_{q,r}^{i}\|$. We define the *average error span*, E_s, of a VLEC code over the BSC as the average number of source symbol errors in an error event, where the average is taken over all error events. Similarly, we define the *average effective error span*, $E_{s_{\text{eff}}}$, of a VLEC code as the average Levenshtein distance $L(\mathbf{a}_i, \mathbf{a}_j)$.

Lemma 5.1: The average effective error span of a VLEC code is given by

$$E_{s_{\text{eff}}} = \frac{P_s(E)}{P(E)} \tag{5.1}$$

where $P(E)$ is the error event probability and $P_s(E)$ is the SEP as calculated using the algorithm given in Section 2.9.2.

Proof: From the algorithm given in Section 2.9.2,

$$P_s(E) = \frac{\sum_{i,j} L(\mathbf{a}_i, \mathbf{a}_j)}{|\mathbf{a}_t|} \tag{5.2}$$

where $\mathbf{a}_i$ and $\mathbf{a}_j$ are, respectively, the correct and incorrect message sub-sequences corresponding to an error event and $\mathbf{a}_t$ is the transmitted message sequence. The sum is taken over all error events. Whereas the error event probability is given by

$$P(E) = \frac{\text{Total number of error events}}{|\mathbf{a}_t|} \tag{5.3}$$

Hence, dividing equation (5.2) by (5.3)

$$\frac{P_s(E)}{P(E)} = \frac{\sum_{i,j} L(\mathbf{a}_i, \mathbf{a}_j)}{\text{Total number of error events}} = E_{s_{\text{eff}}} \tag{5.4}$$

■

Therefore, the synchronisation properties of VLEC codes over the BSC may be determined from the error event and the symbol error probabilities. Unfortunately, the upper bounds on $P_s(E)$ and $P(E)$ cannot be applied to bound $E_{s_{\text{eff}}}$. However, we may obtain a useful relationship for the average effective error span of a VLEC code for very small p. This is achieved in Theorem 5.1. However first we need the following lemma.

Lemma 5.2: The error event probability of a VLEC code at high SNR is approximately given by

$$P(E) \approx A_{d_{\text{free}}} P_{d_{\text{free}}} \tag{5.5}$$

where $A_{d_{\text{free}}}$ is the average number of converging pairs of paths at free distance d_{free}, and $P_{d_{\text{free}}}$ is the probability of decoding a path into another one at distance d_{free} over the BSC.

Proof: The proof is similar to the one given for Corollary 3.1 on page 90 and is thus omitted. In this case, the upper bound on the error event probability given by expression (3.31) must be used. ■

Theorem 5.1: The average effective error span of a VLEC code with free distance d_{free} over the BSC with very small p is approximately given by

$$E_{s_{\text{eff}}} \approx \frac{B_{d_{\text{free}}}}{A_{d_{\text{free}}}} \tag{5.6}$$

where $A_{d_{\text{free}}}$ is the average number of converging pairs of paths at free distance d_{free}, and $B_{d_{\text{free}}}$ is the average pairwise Levenshtein distance of these paths.

Proof: The required relationship is obtained by substituting for $P(E)$ and $P_s(E)$ in equation (5.1) using the approximate relationships given respectively by expressions (5.5) and (3.37). ■

Note that the minimum possible value for $E_{s_{\text{eff}}}$ (and E_s) over the BSC is one symbol, signifying that in an error event there is a single symbol decoded in error (otherwise there would be no error event) but no synchronisation loss. In fact, for fixed-length codes (over the BSC) $E_s = E_{s_{\text{eff}}} = 1$ symbol always.

In order to obtain similar expressions for the average error span, we define the average number of source symbols in all converging pairs of paths whose encoded messages are at a Hamming distance h from each other, C_h, as follows

$$C_h = \sum_{k=1}^{\infty} \sum_{\substack{(i,j)\in G_{0,k} \\ H(p_{0,k}^i, p_{0,k}^j)=h}} \left\| p_{0,k}^i \right\| P\left(p_{0,k}^i\right) \tag{5.7}$$

where $p_{0,k}^i$ is the ith path from state S_0 to state S_k, having $\left\| p_{0,k}^i \right\|$ source symbols and probability $P(p_{0,k}^i)$, $G_{0,k}$ is the set of all pairs of path indices corresponding to paths which diverge at state S_0 and merge again for the first time at state S_k and $H(p_{0,k}^i,\ p_{0,k}^j)$ is the

Hamming distance between the encoded paths $p^i_{0,k}$ and $p^j_{0,k}$. The values for C_h may be determined using the same algorithms as those given in Section 3.5.1.1.

Theorem 5.2: The average error span for a VLEC code with free distance d_{free} over the BSC for very small p is approximately given by

$$E_s \approx \frac{C_{d_{\text{free}}}}{A_{d_{\text{free}}}} \tag{5.8}$$

Proof: The proof is similar to that of Theorem 5.1 and is omitted. ■

The definition for the average error span as $p \to 0$ is the same as that given by Maxted and Robinson [1985] for (non-error-correcting) exhaustive variable-length codes under single-bit errors, since for these codes a single-bit error will always cause an error event. Indeed, for the special case of exhaustive variable-length codes with $d_{\text{free}} = 1$, approximations (5.6) and (5.8) become equalities, since in this case all possible error events are accounted for by paths contributing to $A_{d_{\text{free}}}$. Using (5.8), the same values for E_s as derived by Maxted and Robinson [1985] using state model techniques, for various Huffman codes are obtained. For instance, for code C_{12} given as Code 1 in Table VI in their paper, reproduced here in Table 5.1, A_1=2.200, B_1=3.574 and C_1=3.7625. Hence, using expression (5.8), E_s = 1.7102, which is the same as that calculated in the above mentioned paper.

Symbol	Probability	Code C_{12}
a	0.4	01
b	0.2	00
c	0.2	11
d	0.1	100
e	0.1	101

Table 5.1: Huffman code given in Maxted and Robinson [1985]

As an example of the variation of the average effective error span with the value of the cross-over probability over the BSC, we give in Figure 5.1 that for code C_{17} with free distance five, given in Table A.2, when used to encode the 26-symbol English source. For this code, A_5=2.3076, B_5=3.8543 and C_5=3.6209, hence for small p, $E_{s_{\text{eff}}} \approx 1.67$. We note,

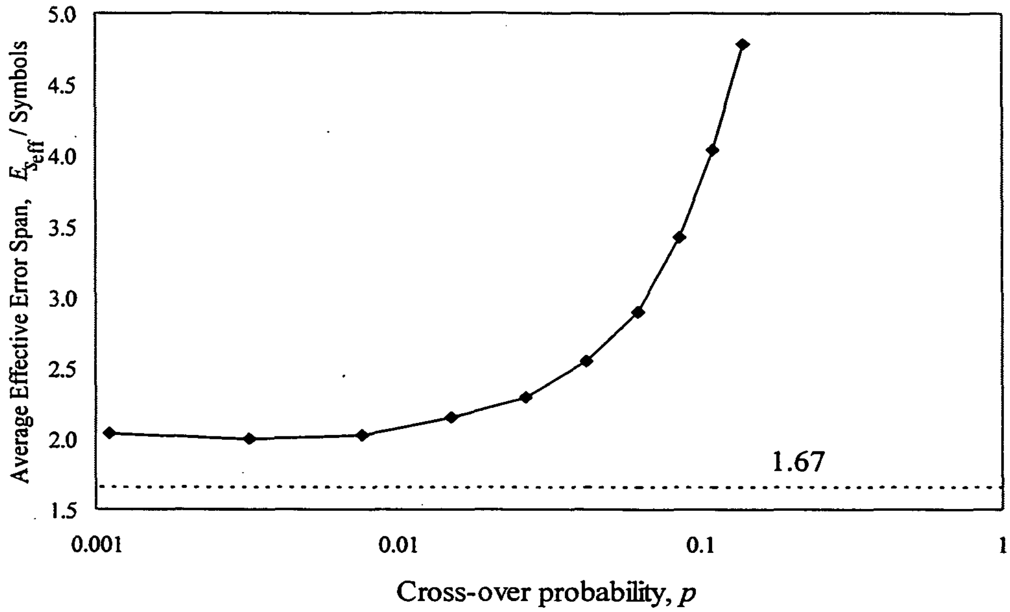

Figure 5.1: Variation of average effective error span with cross-over probability for C_{17}

unfortunately, that the approximation given by (5.6) in this case is not very accurate. This is because we are ignoring paths which are at distance $d_{free} + 1$, which, since in this case d_{free} is odd, have the same probability of being decoded in error as those at distance d_{free}. The interactions between these paths, however, is not easy to analyse.

For all VLEC codes considered, the average (effective) error span increased with increasing p; however, for certain non-error-correcting variable-length codes this behaviour may be inverted. For an example of such a code see Rahman and Misbahuddin [1989].

Care should be exercised when using expressions (5.6) and (5.8) for E_s and $E_{s_{eff}}$. For instance, for code C_3 given in Table 2.3 (with any source), for very small p, $E_s = E_{s_{eff}} = 1$. This of course does not mean that this code is immune to loss of sync. Rather, as p becomes very small, when an error event occurs, with large probability this will be caused by the incorrect decoding of a codeword into one of the same length. This is brought about by the fact that the free distance for this code is three, but the unequal length free distance is five. Hence, the probability that an error event consists of more than one source symbol is negligibly small compared with that for single source symbol error events for very small p. Note, however, that the overall performance of the code depends also on $P(E)$.

5.3. Synchronisation Recovery without Start of Message

When the channel alters the number of received bits from that transmitted, one of the main assumptions for maximum likelihood decoding used in Chapter 3 will no longer be valid, since knowing the number of received bits N will no longer be enough to determine the set of possible paths transmitted. In fact, under this channel behaviour it is certain that the modified Viterbi decoding algorithm for VLEC codes will make a decoding error. However, it will be interesting to investigate the effects of these types of channel errors on the performance of this algorithm.

The first case we are going to consider in this section is that when the initial bits in the transmitted message are lost, such as when the receiver is not initially synchronised with the transmitter and there are no special symbols indicating the start of message. Another example is when the receiver starts decoding in the middle of a message. In what follows we are going to assume that there are no further errors on the channel during the re-synchronisation period.

Suppose that n bits are lost at the start of the message. Since the decoding algorithm does not attempt to correct insertion and/or deletion errors, then we will consider decoding errors only from the first complete codeword received. If the number of bits lost constitute an exact sequence of codewords, then obviously under the assumption that no further errors occur on the channel, there will be no synchronisation loss and hence no decoding errors as far as we are concerned[1]. Hence, without loss of generality, we are going to consider that the n bits lost are contained within a single codeword. Let the length of this codeword be l. If the decoder knows that it is still acquiring synchronisation, then it does not make sense to assign a metric of zero to state S_0 and minus infinity to all the others as required by the modified Viterbi algorithm. Rather, the decoder must assign a metric zero to all initial states, since it does not know which of these corresponds to the first received bit. Starting with this assumption, we can prove the following strong result.

[1] Here we are assuming that the VLEC code itself is uniquely decodable and has a finite decoding delay.

Theorem 5.3: For a non-sequentially catastrophic VLEC code C ($s_1@L_1$, b_1; $s_2@L_2$, b_2; $\cdots$; $s_\sigma@L_\sigma$, b_σ; $d_{\min}$, $c_{\min}$) with $\gcd(L_1, L_2, \cdots, L_\sigma) = 1$ and $c_{\min} \neq 0$, the modified Viterbi algorithm (with all initial path metrics set to zero) will synchronise on the first complete codeword received, provided that there are no further channel errors.

Proof: Suppose that the number of bits lost from the first incomplete codeword received is n, where $0 \leq n < L_\sigma$. Note that if $n = 0$, the first bits received will in fact form a complete codeword. Now, if n is not a multiple of $\gcd(L_1, L_2, \cdots, L_\sigma)$, then the received sequence is in a state which does not exist in the trellis. Hence in this case the decoder will never synchronise[2]. Therefore, in order to ensure that the received sequence is in a state which does exist for any value of n, we need that $\gcd(L_1, L_2, \cdots, L_\sigma) = 1$.

Now, suppose that the decoder will never synchronise with the original message as shown in Figure 5.2. It is given that both the received and the decoded sequences are assigned an initial metric of zero. Furthermore, since we are assuming that there are no further channel errors, then the final metric on the correct path will be zero. Hence, the only way a maximum likelihood decoder for VLEC code will make an incorrect decoding is when the incorrect path also has a final metric of zero. If the two paths are assumed to be infinitely long, then this could only happen if the two paths, when padded with a finite number of appropriate codewords at both ends so as to make them of equal length, are at a finite distance from each other. However, since the code is non-sequentially catastrophic

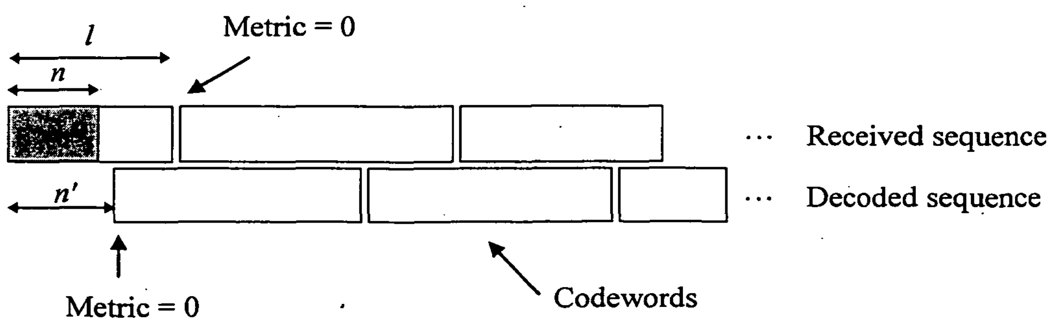

Figure 5.2: Incorrect synchronisation after n initial bits lost

[2] This is the case, for instance, with fixed-length codes, where, if the number of lost bits is not a multiple of the codeword length, a maximum likelihood decoder will never synchronise by itself.

this is impossible, so at some point the decoder must re-synchronise. So, at least, the decoder has a finite synchronisation delay. However, at the point of synchronisation, the final two codewords in the two paths must be at least distance $c_{\min}$ from each other. Therefore for non-zero $c_{\min}$, the out-of-sync path cannot have a metric zero, and hence the decoder will choose the correct path and will synchronise on the first complete codeword received. ■

If the decoder does not know that synchronisation has not been acquired and tries to normally decode the received message when n bits are lost at the start of the message, then the initial metrics assigned to the trellis' states are as given in the modified Viterbi algorithm in Section 3.2.2.2, i.e. state S_0 assigned metric 0 and all other states minus infinity. In this case Theorem 5.3 does not hold and the synchronisation delay is longer. Figure 5.3 gives the average effective error span for code C_{17} given in Table A.2 when used to encode the 26-symbol English source, under this condition, for various number of initial bits lost, obtained using computer simulation. Note that in calculating the average effective error span, all deleted or partially corrupted message source symbols are not counted, however, all decoded source symbols are considered, even those relative to the possibly partially received codeword at the beginning of the message. From this figure we may note

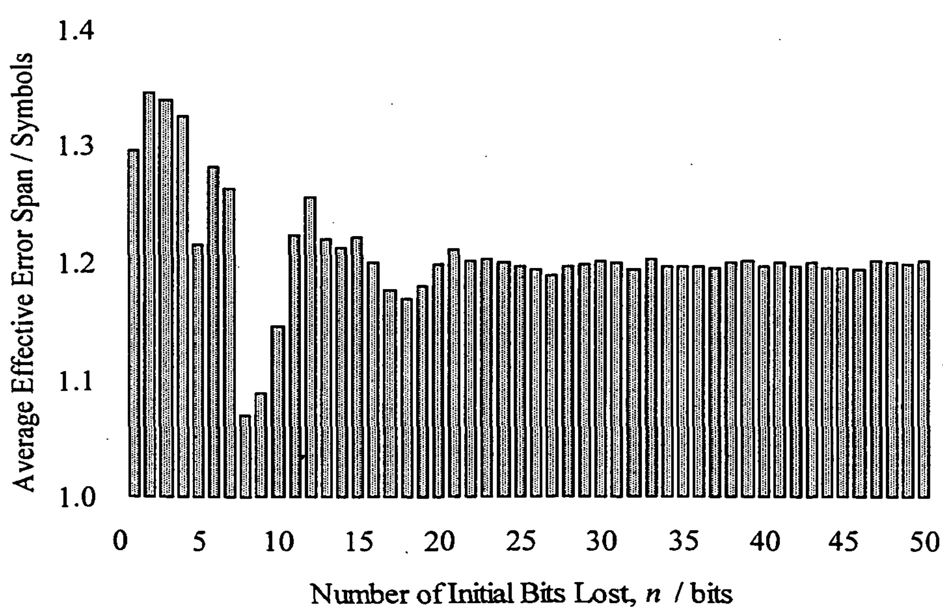

Figure 5.3: Effective error span for code C_{17} with normally assigned initial path metrics

that starting from a random position from within the message results in an average effective error span of just 1.2 symbols. In this case the synchronisation process is similar to when bits are lost within the message. Hence an explanation of the synchronisation mechanism will be deferred to the next section.

5.4. Synchronisation Recovery on Channels with Symbol Deletions and Insertions

In the previous section we considered the special case when the decoder is initially out of synchronisation. A more general model for the channel is one which allows channel symbol deletions and insertions. As was discussed in Chapter 2, these kinds of errors have a disastrous effect on fixed-length error-correcting block codes. They could also present problems in the case of convolutional codes, however here the decoder can recover synchronisation with minimal increase in complexity using some clever techniques (see for example Sodha and Tait [1992]). In any case, however, for both block and convolutional error-correcting codes the decoder must be modified so as to be able to recover synchronisation after a channel symbol deletion or insertion. As we shall show empirically here, this is not necessary in the case of VLEC codes.

5.4.1. Symbol Deletions

Figure 5.4 shows the variation of the average effective error span with the number of consecutive bits deleted after 200 bits are received, for the codes C_{15}, C_{16} and C_{17} given in Table A.2 when used to encode the 26-symbol English source. This was obtained using computer simulation. Again note that the average effective error span includes all the incorrectly decoded symbols, including the ones relative to incomplete codewords received, but not the message source symbols which are not completely received. Notice that the maximum average effective error span is just over 1.6 symbols. Considering the fact that these codes and the decoding algorithm were not designed to combat deletion errors, their performance is extremely good. Surprisingly, the performance of the

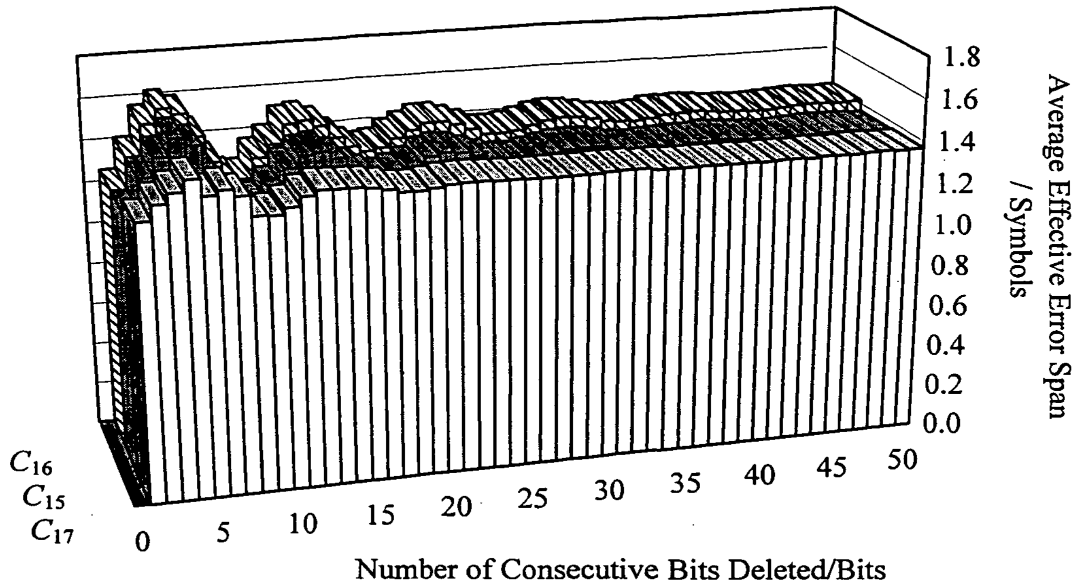

Figure 5.4: Performance for codes C_{15}, C_{16}, and C_{17} for a number of consecutive bits deleted after bit position 200

catastrophic code C_{15} is as good as that for C_{17}, which has a good CDF growth rate (see Chapter 4).

The re-synchronisation process for a VLEC code C ($s_1@L_1$, b_1; $s_2@L_2$, b_2; $\cdots$; $s_\sigma@L_\sigma$, b_σ; $d_{\min}$, $c_{\min}$) with a maximum likelihood decoder may be explained as follows. Consider first a single bit deleted from a codeword of length l, resulting in a received word $\mathbf{w}_d$ of length $l-1$. If the $\gcd(L_1, L_2, \cdots, L_\sigma) \neq 1$, then it is impossible for the decoder to regain synchronisation, as was explained in the previous section. Hence, for automatic re-synchronisation it is required that $\gcd(L_1, L_2, \cdots, L_\sigma) = 1$. If $l-1 = L_i$, for some $i = 1, 2, \cdots, \sigma$, then the decoder may decode $\mathbf{w}_d$ into a single codeword $\mathbf{c}_d$, with $|\mathbf{c}_d| = l-1$, resulting in no synchronisation loss, as shown in case (a) in Figure 5.5. Again assuming that there are no further errors on the channel, for this to occur we need that $H(\mathbf{w}_d, \mathbf{c}_d)$ be smaller than the distance of all incorrectly synchronised merged paths to the received one, an example of which is shown in case (b) in Figure 5.5. Unfortunately we cannot say much about the distance of a sequence like that of case (b) to the received sequence, other than that it must be at least distance $c_{\min}$. To ensure that $\mathbf{w}_d$ is some minimum diverging distance to all codewords of C will be too large a constraint which will result in low rate codes. For a

sequentially catastrophic code it may also be possible to find an infinitely long unmerged path to the received sequence which is at finite distance. If this distance is less than $H(\mathbf{w}_d, \mathbf{c}_d)$, this means that the decoder will never re-synchronise. Clearly, therefore, if synchronisation is important, we should avoid using sequentially catastrophic codes. Conversely, if the code is not sequentially catastrophic, then the synchronisation delay will be bounded. Case (b) will also arise if there are no codewords of length $l-1$. Hence, it is advantageous to have consecutive codeword lengths, i.e. $L_1 = L_2-1 = \cdots = L_\sigma - \sigma + 1$. Also, the larger σ is, the more probable that a codeword of length $l-n$ exists. Therefore for good synchronisation properties σ must be chosen as large as possible.

The re-synchronisation mechanism described above is also true when the number of deleted bits is greater than one, except that the probability that the corrupted word has length not equal to any codeword length increases. In addition, another effect starts to become important. When two or more bits are deleted, the probability that two adjacent codewords are affected increases. Hence we would expect that the average effective error span increases as the number of deleted bits increases. However, when L_1 or more bits are deleted, there is the probability that the deleted bits constitute a complete codeword, again ensuring that there is no synchronisation loss. In addition, the actual code construction has a direct influence on the synchronisation behaviour. Notice from Figure 5.4 that initially the effective error span increases with increasing number of bits deleted, until around the

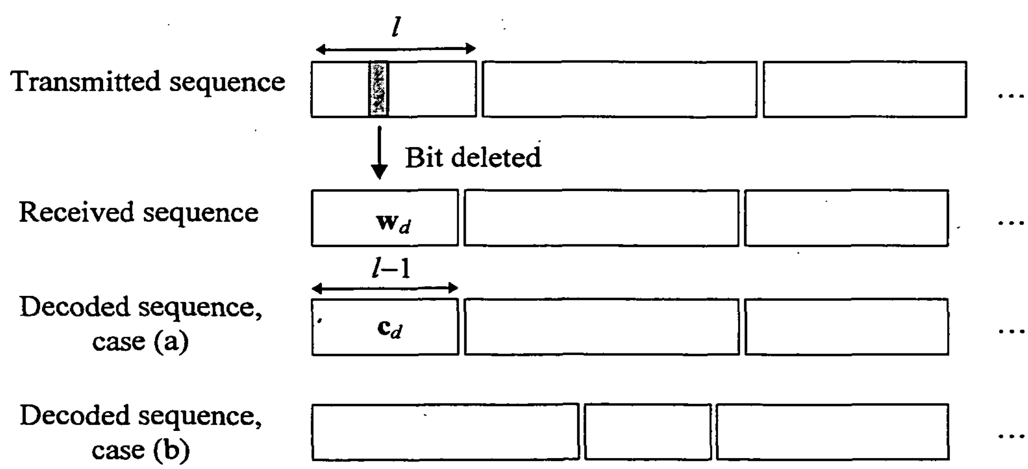

Figure 5.5: Synchronisation recovery under a single bit deletion

minimum codeword length it starts to decrease again. This will reach a minimum when the number of bits deleted is approximately equal to the average codeword length, which for codes C_{15}, C_{16} and C_{17} is approximately 8.5 bits, because the probability of deleting a complete codeword is highest in this case.

Figure 5.6 shows the probability distribution of the effective error spans for codes C_{15}, C_{16} and C_{17} when four consecutive bits are deleted after bit position 200, which for all three codes results in the maximum average effective error span (see Figure 5.4).

As an example of the effect of σ on the effective error span, consider Figure 5.7 which gives the effective error span for codes C_{18} and C_{19} given in Table A.2 when used to encode the 26-symbol English source. For C_{18}, $\sigma = 3$ while for C_{19}, $\sigma = 5$. Notice the large difference in performance between these two codes and even between these and the previous codes considered, for which $\sigma = 8$, 8 and 11 respectively for C_{15}, C_{16} and C_{17}.

5.4.2. Symbol Insertions

Channel symbol insertion is complementary to channel symbol deletion, whereby one or more random bits may be inserted within the transmitted message. The re-

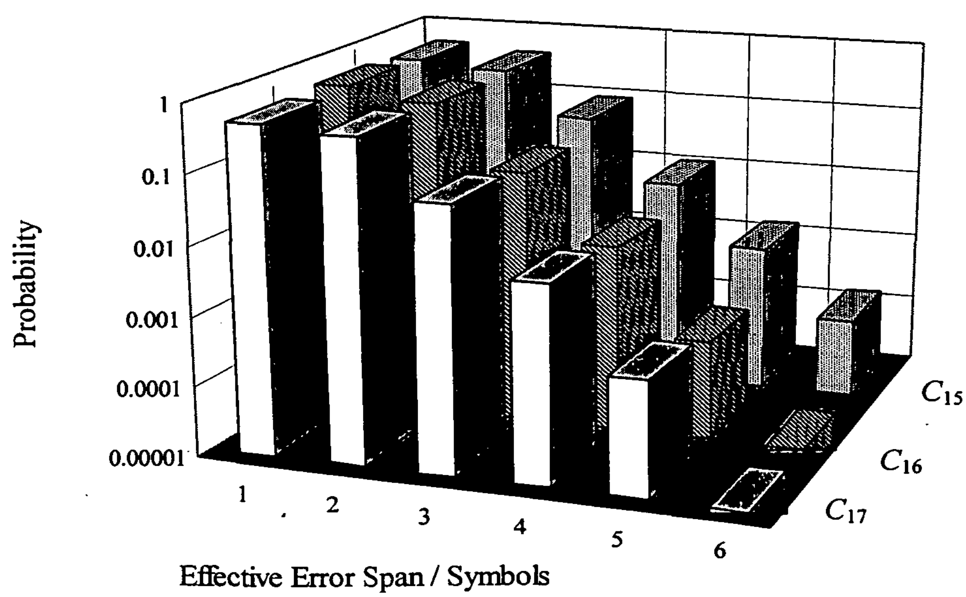

Figure 5.6: Probability distribution of the effective error span for codes C_{15}, C_{16}, and C_{17}

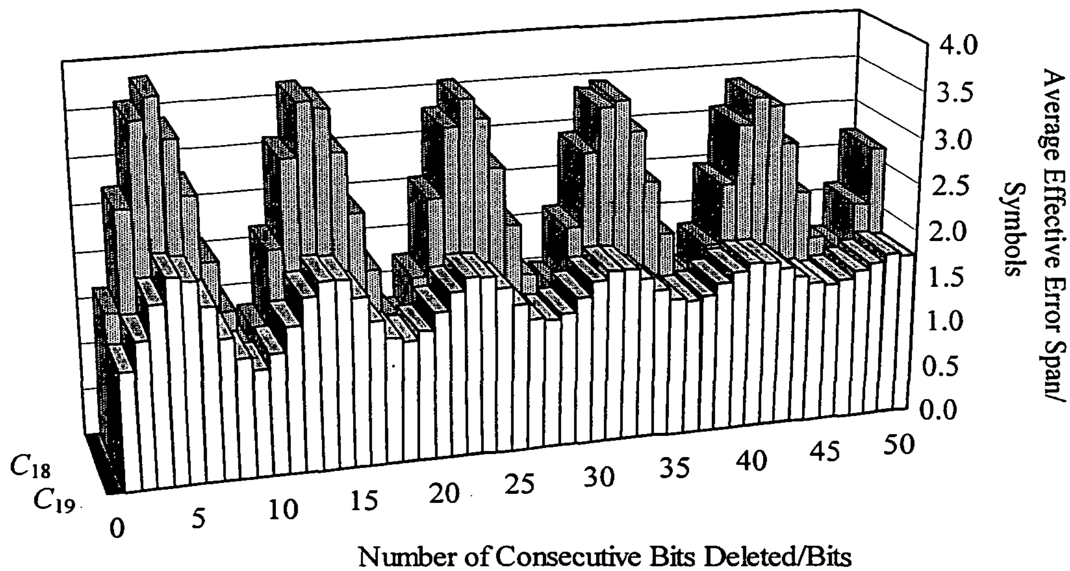

Figure 5.7: Performance for codes C_{18} and C_{19} for a number of consecutive bits deleted after bit position 200

synchronisation process is very similar to that described in the previous section for symbol deletion. Figure 5.8 shows the average error span for the three codes C_{15}, C_{16}, and C_{17} when used to encode the 26-symbol English source, as it varies with the number of consecutively inserted random bits after bit position 200 within the transmitted message. In this case it was considered to be more appropriate to measure the average error span, since the number of erroneously decoded symbols will increase with the number of inserted bits, whereas the number of complete source symbols received is at most one less than that transmitted, whatever the number of (consecutively) inserted bits. Again, the minimum value for the average error span occurs when the number of inserted bits is approximately equal to the average codeword length.

5.5. Conclusion

Theorem 5.3 shows that the synchronisation performance of non-sequentially catastrophic VLEC codes with codeword lengths having no common factor and with a non-zero minimum converging distance is excellent when the decoder knows that it is out of sync, since in this case it will synchronise on the first complete codeword received

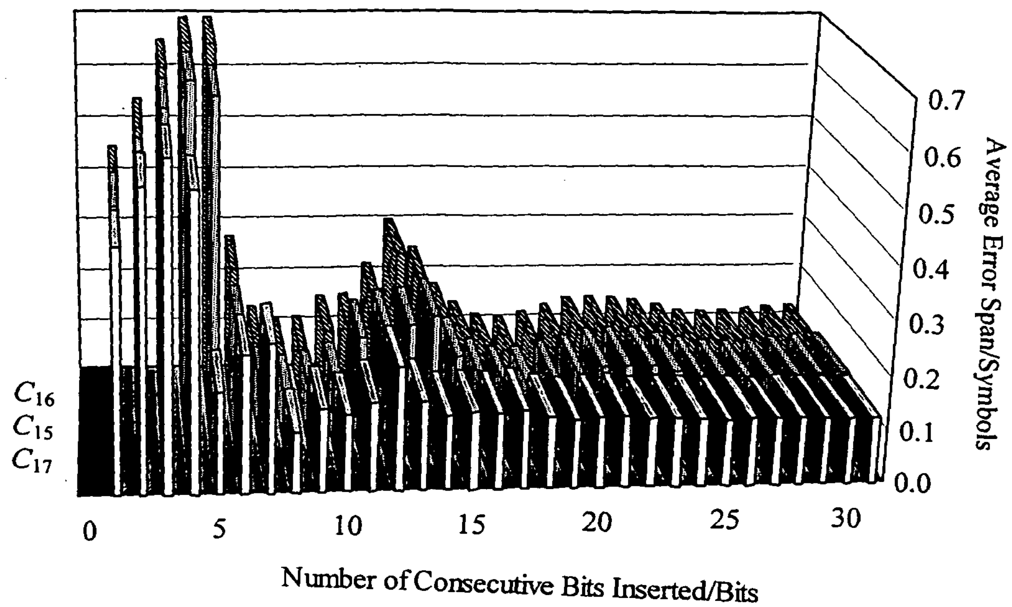

Figure 5.8: Performance for codes C_{15}, C_{16}, and C_{17} for a number of consecutive random bits inserted after bit position 200

(provided that the received sequence has an appropriately long period without further errors). In addition, even when the decoder does not know that there is loss of synchronisation, the average error span can be kept quite small. For the codes considered here, the maximum average effective error span was found to be only 1.6 source symbols. This occurred under bit deletion errors. This figure can be kept small by designing the code to have as many consecutive codeword lengths with no common factor as possible. Note that this will also allow us to construct good VLEC codes with codeword lengths matching the source statistics. Other constraints could be imposed on the VLEC codes to improve their synchronisation properties, however we feel that these will drastically increase the average codeword length, to be effective, making them impractical.

From the results presented in this chapter, we have seen that in practice there is not much difference in the synchronisation performance of sequentially catastrophic and non-sequentially catastrophic VLEC codes. However, in the case of sequentially catastrophic VLEC codes, for particular messages under specific error patterns, the decoder may never achieve synchronisation. But in most cases the probability for these to occur is very small,

and hence these do not affect the performance of the code. The performance penalty for sequentially catastrophic codes is even less than for strictly catastrophic codes as defined in Section 3.4.3, because in this case, although synchronisation may be delayed, the decoder may still be decoding correctly even when out of synchronisation.

Designing the unequal length free distance of a VLEC code to be greater than its free distance will make it more robust against loss of synchronisation, since the minimum block distance of a code does not influence its synchronisation properties. Hence, when synchronisation is of prime concern, it is advisable to make the unequal length free distance greater than the free distance. However we do not see this as advantageous as regards to the SEP performance, especially in the case of the AWGN channel. Recall from Chapter 3 that the SEP performance of a VLEC code over the AWGN channel with hard decision maximum likelihood decoding is very much dependent on its free distance (not the unequal free distance). In order to increase the unequal free distance of a code, its average codeword length must correspondingly increase. This will offset any gain achieved through increase in the unequal free distance, since this has minimal effect.

Dunscombe [1988] and Escott [1995] have shown that two-length VLEC codes with codeword lengths L and $2L$ have very good synchronisation properties over the BSC. However, there are two problems with these. First, two-length codes are only one step away from fixed-length codes, which have perfect synchronisation properties over the BSC. In order to really match codeword lengths to the source statistics to achieve the minimum codeword length possible for a given free distance, we must allow any codeword length. In this case the average error span may increase, but the error event probability (for high SNR) will be lower than for the two-length codes, giving an overall improvement. A comparison between the performance of these codes over the AWGN channel with hard decision decoding will be presented in Chapter 6. The second problem, arising on channels which allow symbol insertion and/or deletion, is more serious, since in this case the performance of these codes is as bad as for fixed-length codes, in that the GCD of their codeword lengths is not equal to one. Hence for two-length codes a decoder may never recover synchronisation by itself over these channels.

Chapter 6.

Code Constructions

6.1. Introduction

In the previous chapters, we have defined some properties of VLEC codes relating to their performance and characteristics. However, with the exception of α-prompt codes discussed in Chapter 2, we have deferred any mention of how to actually construct such codes until this chapter.

As with standard error-correcting codes, we can sub-divide VLEC codes into linear and non-linear codes. It turns out that a VLEC code can never be entirely linear. However, by suitably sub-dividing a code, linear sub-structures may be defined. Linearity is important because

- it allows simpler encoding and decoding algorithms to be used, by exploiting the additional mathematical structure;
- it simplifies code construction, first by again utilising mathematical structures and secondly by reducing the possible domain from where the code is chosen.

We will basically present two construction techniques for VLEC codes. The first, presented in Section 6.3, uses fixed-length linear codes and anticodes to build new VLEC codes, whereas the second, given in Section 6.4, uses a heuristic algorithm to perform a computer search for good VLEC codes.

Finally, the performance of some VLEC codes is compared to that of standard schemes for source and error-control coding.

6.2. Linear VLEC Codes

Theorem 6.1: A non-trivial, uniquely decodable VLEC code is always non-linear.

Proof: Let $\mathbf{c}_i$ and $\mathbf{c}_j$ be two unequal length codewords of a VLEC code C. Note that we can always find two such codewords in any non-trivial VLEC code C, since C must at least have two codewords and for it to be classified as variable-length, these must necessarily be of unequal length. Let $|\mathbf{c}_i|=l_i$ and $|\mathbf{c}_j|=l_j$. Then, for the code to be linear, $\mathbf{c}_i+\mathbf{c}_i=\mathbf{0}_{l_i}$ and $\mathbf{c}_j+\mathbf{c}_j=\mathbf{0}_{l_j}$ must both be codewords in C, where $\mathbf{0}_l$ represents the word with l all zero bits. However, if both $\mathbf{0}_{l_i}$ and $\mathbf{0}_{l_j}$ are codewords of C, then C is not uniquely decodable. This can be shown to be true by considering the codeword sequence consisting of l_j consecutive $\mathbf{0}_{l_i}$ which is indistinguishable from the codeword sequence consisting of l_i consecutive $\mathbf{0}_{l_j}$. Hence a uniquely decodable VLEC code can never be linear. ■

Theorem 6.1 does not exclude us, however, from using linear sub-codes, or cosets of linear sub-codes, within the VLEC codes. It does state, however, that these sub-codes must be of fixed length. Thus, we can have two different linear structures imposed on VLEC codes, called respectively *horizontal* and *vertical linearity*, shown diagramatically in Figure 6.1.

Definition 6.1: Define the vertical sub-codes of a VLEC code C ($s_1@L_1$, b_1; $s_2@L_2$, b_2; $\cdots$; $s_\sigma@L_\sigma$, b_σ; $d_{\min}$, $c_{\min}$), with $C = \{\mathbf{c}_1, \mathbf{c}_2, \cdots, \mathbf{c}_s\}$, to be $V_1, V_2, \cdots, V_\sigma$, where $V_i = \{\mathbf{c}_{\bar{s}_i+1}^{L_{i-1}+1,L_i}, \mathbf{c}_{\bar{s}_i+2}^{L_{i-1}+1,L_i}, \cdots, \mathbf{c}_s^{L_{i-1}+1,L_i}\}$, $i = 1, 2, \cdots, \sigma$, L_0 is defined to be equal to 0 and $\mathbf{c}_j^{a,b} = c_{j_a}c_{j_{a+1}}\cdots c_{j_b}$ given that $\mathbf{c}_j = c_{j_1}c_{j_2}\cdots c_{j_{l_j}}$, $\mathbf{c}_j \in C$. Then, a VLEC code C is said to be *vertically linear* iff its vertical sub-codes $V_1, V_2, \cdots, V_\sigma$ are all cosets of linear fixed-length codes.

Definition 6.2: Define the horizontal sub-codes of a VLEC code C ($s_1@L_1$, b_1; $s_2@L_2$, b_2; $\cdots$; $s_\sigma@L_\sigma$, b_σ; $d_{\min}$, $c_{\min}$), with $C = \{\mathbf{c}_1, \mathbf{c}_2, \cdots, \mathbf{c}_s\}$, to be $H_1, H_2, \cdots, H_\sigma$, where $H_i = \{\mathbf{c}_{\bar{s}_i+1}, \mathbf{c}_{\bar{s}_i+2}, \cdots, \mathbf{c}_{\bar{s}_i+s_i}\}$, $i = 1, 2, \cdots, \sigma$. Then, a VLEC code C is said to be *horizontally linear* iff its horizontal sub-codes $H_1, H_2, \cdots, H_\sigma$ are all cosets of linear fixed-length codes.

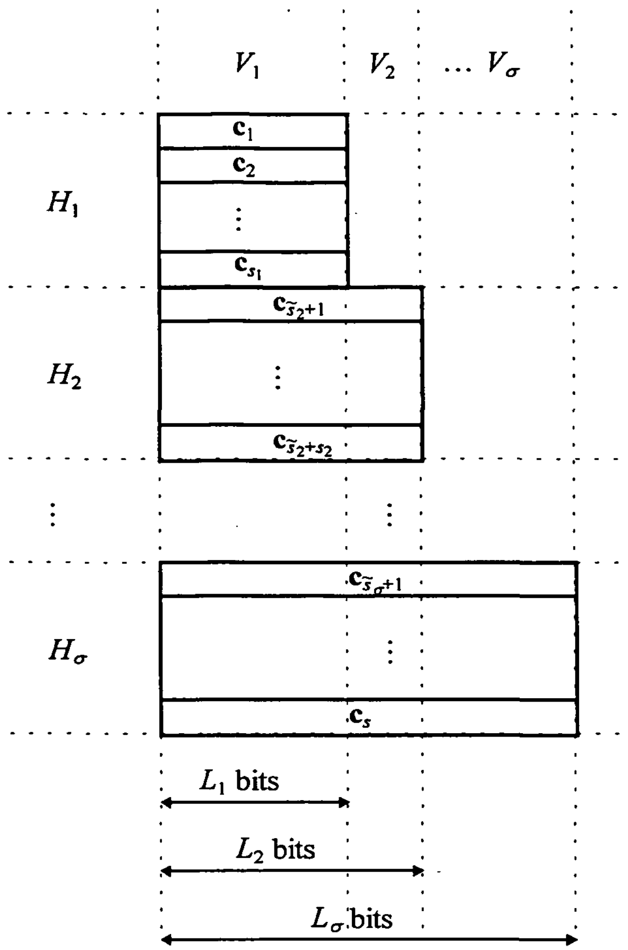

Figure 6.1: Horizontal and vertical sub-codes of a VLEC code

6.2.1. Vertically Linear VLEC Codes

The vertical sub-codes V_1, V_2, $\cdots$, V_σ as defined above are the same as the segment decomposition of C, $\{Q_1, Q_2, \cdots, Q_\sigma\}$ as given in Section 2.6.2, i.e. $V_i = Q_i$, i=1, 2, $\cdots$, σ. Hence it is natural that vertically linear VLEC codes are well suited to be decoded using the segment decoding algorithm given in Section 2.6.3. It is interesting to note that in any vertical sub-code we may have repeated codewords. This is not normal for an ordinary fixed-length code. Here, however, the ambiguity between the repeated codewords is resolved by the other codeword segments in the other vertical sub-codes.

One of the reasons for using a linear code is that it facilitates the decoding process. Now, we have seen that for good coding gain we must decode VLEC codes using either

maximum likelihood decoding (Chapter 3) or sequential decoding (Chapter 4). In both these algorithms, the parallel transitions must be resolved immediately. Hence, in order to simplify the decoding process for both these algorithms, it is advantageous to have a linear structure on the parallel transitions. Vertical linearity does not help much in this respect because it does not simply resolve which transition to decode for each group of parallel transitions. For this reason, we shall not discuss any further vertically linear VLEC codes.

6.2.2. Horizontally Linear VLEC Codes

If a VLEC code C ($s_1@L_1$, b_1; $s_2@L_2$, b_2; $\cdots$; $s_\sigma@L_\sigma$, b_σ; $d_{\min}$, $c_{\min}$) is horizontally linear with horizontal sub-codes H_1, H_2, $\cdots$, H_σ, the modified Viterbi decoding algorithm given in Section 3.2.2.2 may be simplified as follows. Instead of computing the branch metric for each codeword in C (Step 2), H_1, H_2, $\cdots$, H_σ are used to determine the σ most likely codewords for each respective possible codeword length. The metrics for these σ codewords are then computed normally and these will then be the codewords stored for the transitions $S_i \rightarrow S_{i+L_j}$, $j=1, 2, \cdots, \sigma$. Hence, instead of computing the metric for s codewords, this is now computed for σ codewords. In addition, of course, we now have to have σ linear decoders to determine the most likely codeword for each possible length. Similarly, in the stack decoding algorithm for VLEC codes given in Section 4.3, H_1, H_2, $\cdots$, H_σ are used to determine the σ most likely codewords (of different lengths). These will then be used in Step 2 to extend the path at the top of the stack. Hence in this case, only σ new paths are generated instead of s and Step 3 becomes redundant.

It is obvious from the definition of H_1, H_2, $\cdots$, H_σ, that the minimum block distance b_i corresponding to the codeword length L_i will be equal to the minimum distance of H_i. Hence, if the overall minimum block distance is $b_{\min}$, then the minimum distance for each one of the horizontal sub-codes must at least be equal to or greater than $b_{\min}$. This is a relatively easy requirement to satisfy, since for a given number of codewords and block length we can determine, using tables, the minimum distance achievable using a fixed-length linear code. However, we also require that the various horizontal sub-codes must have a certain distance from each other, determined by the minimum diverging and

converging distances, $d_{\min}$ and $c_{\min}$ respectively. The construction algorithm given in the next section splits a fixed-length code into cosets with specified minimum distances and then uses an anticode [Farrell, 1970] [Farrell & Farrag, 1974] [Farrell, 1977] to remove certain columns of some of these cosets. This guarantees an overall minimum block distance and a minimum diverging distance, however the minimum converging distance necessary to give the required free distance may not always be satisfied.

6.3. Code-Anticode Construction

Given that we want to construct a binary horizontally linear VLEC code with s codewords and free distance d_{free} we proceed as follows (*code-anticode construction*)

1. Find a binary linear fixed-length code $\mathcal{F}$ with parameters (n, k, d) with minimum n such that $2^k \geq s$ and $d = d_{\text{free}}$, where n is the block length, 2^k is the number of codewords and d the minimum distance.
2. Let $d_{\min} = \lceil d_{\text{free}}/2 \rceil$.
3. Rearrange the columns of $\mathcal{F}$ such that the rightmost m columns form an anticode $\mathcal{A}$ with parameters (m, k, δ) with maximum m for $\delta = d_{\text{free}} - d_{\min}$, where m is the block length, 2^k is the number of codewords, and δ is the maximum distance for the anticode.
4. Re-order the codewords of $\mathcal{F}$ such that repeated codewords of $\mathcal{A}$ are consecutive.
5. Perform simple row operations and column permutations on the generator matrix for $\mathcal{F}$ such that the generator matrix of $\mathcal{A}$ contains the maximum number of consecutive 0's in the top most positions of each column, starting from the rightmost column.
6. Delete the rightmost m columns of the first s_1 codewords in $\mathcal{F}$, where s_1 is the number of codewords with identical m rightmost columns.
7. Considering the remaining codewords in $\mathcal{F}$, delete the rightmost $m-1$ columns of the next s_2 codewords in $\mathcal{F}$, where s_2 is the number of codewords with identical $m-1$ rightmost columns.
8. Repeat Step 7 with the next $m-2$, $m-3$, ... rightmost columns until there are no further columns to delete.

Note that Step 5 will ensure that the maximum number of codewords with the same rightmost columns are generated, hence resulting in a VLEC code with a shorter average codeword length.

Theorem 6.2: The VLEC code constructed using the code-anticode construction from the fixed-length code with parameters (n, k, d) and the anticode with parameters (m, k, δ) is a horizontally linear VLEC code with overall minimum block distance at least equal to d and minimum diverging distance at least equal to $d - \delta$.

Proof: Step 1 in the code-anticode construction ensures that all codewords are at least distance d from each other. Now, by deleting the rightmost m columns in Step 6 of the first s_1 codewords in $\mathcal{F}$ results in two sub-codes, one of length $n - m$ with s_1 codewords (sub-code H_1) and one of length n with $s - s_1$ codewords (sub-code $H_2^{'}$). Since the deleted columns from H_1 are identical (Step 6), then the codewords of H_1 form a linear code with minimum distance at least d. Also, since $H_2^{'}$ is a sub-code of the original code $\mathcal{F}$ (with the same length), then it also has minimum distance at least d. Hence the overall minimum block distance of the VLEC code $\{H_1, H_2^{'}\}$ is at least d. In addition, since the m rightmost columns of $\mathcal{F}$ form an anticode with maximum distance δ, then by deleting the m rightmost columns from $\mathcal{F}$ will result in a code with minimum distance $d - \delta$. Hence, the minimum diverging distance between the codewords of H_1 and $H_2^{'}$ must at least be $d - \delta$. Exactly the same reasoning applies for all the other columns deleted. ∎

Unfortunately, as one may observe from Theorem 6.2, this construction algorithm does not guarantee a minimum converging distance. Since with this construction $d_{\min} < b_{\min}$, then using the bound given by expression (3.18) we require $c_{\min} = b_{\min} - d_{\min}$ such that $d_{\text{free}} \geq b_{\min}$. The required value of $c_{\min}$ for a given free distance may be obtained by adding a suitable modification vector to the code (see Section 2.4.2.1) and/or performing column permutations in such a way as not to affect the values of $b_{\min}$ and $d_{\min}$. In general there are 2^{L_σ} possible modification vectors each of length L_σ to test, since adding a modification vector of length L_σ to each codeword in C may affect the minimum

converging distance but not the minimum block and diverging distances[1]. On the other hand, the total number of column permutations possible without affecting $d_{\min}$ and $b_{\min}$ is $\prod_{i=1}^{\sigma}(L_i - L_{i-1})!$, since only columns within a vertical sub-code may be permuted. However, even after performing these operations, the required converging distance still may not be reached. In this case the only alternative is to increase the design value of $d_{\min}$ (in Step 2) and repeat the construction. This in general will decrease the maximum value of m possible for $\mathcal{A}$, resulting in a longer average codeword length for C. Now, however, the required value of $c_{\min}$ is likewise decreased, making it easier to find a suitable modification vector or column permutations. Note that in most cases the total number of column permutations possible may be too large to test all, and in practice it becomes more feasible to find a modification vector instead. We will now illustrate the code-anticode construction with an example.

Example 6.1: Consider that we want to construct a horizontally linear VLEC code for the 26-symbol English source with free distance five. Therefore since $s = 26$, then we must choose $k = 5$. An optimum fixed-length linear code with $k = 5$ and minimum distance five has block length $n = 13$. The generator matrix for this code is shown in Figure 6.2.

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix}$$

Figure 6.2: Generator matrix for (13,5,5) fixed-length linear block code

From Step 2, the design minimum diverging distance is $\lceil \frac{5}{2} \rceil = 3$. Hence, we need to find an anticode with maximum distance 5–3=2. The maximum value for m possible in this case is 3. This is achieved by choosing two columns of $\mathcal{F}$ and their modulo-2 sum as the third column [Farrell, 1977]. Three such columns in $\mathcal{F}$ are columns 2, 3 and 13, where

[1] Note that when adding the modification vector to a shorter length codeword, only the corresponding number of bits from the leftmost position in the modification vector are considered.

the leftmost one is column 1. Hence, rearranging these columns to be in the rightmost positions, and performing simple row operations on the generator matrix of $\mathcal{F}$ so as to arrange its codewords to satisfy Steps 4 and 5, we get the generator matrix given in Figure 6.3. The relative codebook is shown in Figure 6.4.

$$\begin{array}{c} \text{Anticode} \\ \left[\begin{array}{cccccccccc:ccc} 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{array}\right] \end{array}$$

Figure 6.3: Rearranged generator matrix for the (13,5,5) fixed-length linear block code with (3,5,2) anticode in the rightmost position

By deleting the columns indicated by Steps 6-8 in the code-anticode construction, shown shaded in Figure 6.4, we obtain an (8@10,5; 8@11,5; 16@12,5; 3,1) VLEC code, C_{13}. Since the minimum converging distance in this case is one, then the free distance may be less than five (using the bound given by expression (3.18)). In fact, for this code the free distance is four. In order to increase the converging distance without affecting the minimum diverging distance and the overall minimum block distance we can either add an appropriate modification vector to all the codewords in the code, or else perform column permutations. In this example there are 4,096 possible modification vectors and 3,628,800 different column permutations. In fact, adding the modification vector 1111100000000 to code C_{13} will give the required minimum converging distance of two and hence a free distance of five. Figure 6.5 gives the resultant code, C_{14}, plus the required generator matrices and modification vectors for the constituent horizontally linear sub-codes.

The encoding process is likewise simpler, because now the source symbol to be encoded determines which sub-code to use and the corresponding information bits are generated ($\mathbf{k}_1$, $\mathbf{k}_2$ or $\mathbf{k}_3$). These will then be used to calculate the codeword using the respective generator matrix and modification vector.

0 0 0 0 0 0 0 0 0 0	0 0 0
1 0 0 1 0 1 0 0 1 1	0 0 0
0 1 0 0 1 1 1 0 1 0	0 0 0
1 1 0 1 1 0 1 0 0 1	0 0 0
0 0 1 1 1 0 0 1 1 0	0 0 0
1 0 1 0 1 1 0 1 0 1	0 0 0
0 1 1 1 0 1 1 1 0 0	0 0 0
1 1 1 0 0 0 1 1 1 1	0 0 0
0 0 0 0 0 0 1 1 1 0 1	1 0
1 0 0 1 0 1 1 1 0 1 1	1 0
0 1 0 0 1 1 0 1 0 0 1	1 0
1 1 0 1 1 0 0 1 1 1 1	1 0
0 0 1 1 1 0 1 0 0 0 1	1 0
1 0 1 0 1 1 1 0 1 1 1	1 0
0 1 1 1 0 1 0 0 1 0 1	1 0
1 1 1 0 0 0 0 0 0 1 1	1 0
0 0 0 1 1 1 0 0 0 0 0 1	1
1 0 0 0 1 0 0 0 1 1 0 1	1
0 1 0 1 0 0 1 0 1 0 0 1	1
1 1 0 0 0 1 1 0 0 1 0 1	1
0 0 1 0 0 1 0 1 1 0 0 1	1
1 0 1 1 0 0 0 1 0 1 0 1	1
0 1 1 0 1 0 1 1 0 0 0 1	1
1 1 1 1 1 1 1 1 1 1 0 1	1
0 0 0 1 1 1 1 1 1 0 1 0	1
1 0 0 0 1 0 1 1 0 1 1 0	1
0 1 0 1 0 0 0 1 0 0 1 0	1
1 1 0 0 0 1 0 1 1 1 1 0	1
0 0 1 0 0 1 1 0 0 0 1 0	1
1 0 1 1 0 0 1 0 1 1 1 0	1
0 1 1 0 1 0 0 0 1 0 1 0	1
1 1 1 1 1 1 0 0 0 1 1 0	1

Figure 6.4: Codebook for (13,5,5) code and the derived VLEC code, C_{13}

The main disadvantage of this construction algorithm is that the codeword lengths are not matched to the source statistics. In fact the only thing that can be done to reduce the average codeword length in this case is to assign the shorter codewords to the more probable source symbols. Using this mapping, code C_{14} when used to encode the 26-symbol English source given in Table A.1 gives an average codeword length of 10.46 bits. The non-optimality of such a construction may be deduced from the fact that if the shortest length codewords are rearranged such that the last column contains consecutive 0's and 1's and if these 0's are deleted, this will give the VLEC code (4@9,5; 4@10,5; 8@11,5;

$$H_1 \quad \begin{array}{c} 1\,1\,1\,1\,1\,0\,0\,0\,0\,0 \\ 0\,1\,1\,0\,1\,1\,0\,0\,1\,1 \\ 1\,0\,1\,1\,0\,1\,1\,0\,1\,0 \\ 0\,0\,1\,0\,0\,0\,1\,0\,0\,1 \\ 1\,1\,0\,0\,0\,0\,0\,1\,1\,0 \\ 0\,1\,0\,1\,0\,1\,0\,1\,0\,1 \\ 1\,0\,0\,0\,1\,1\,1\,1\,0\,0 \\ 0\,0\,0\,1\,1\,0\,1\,1\,1\,1 \end{array} \qquad \begin{bmatrix} \mathbf{k}_1 \end{bmatrix} \begin{bmatrix} \\ \mathbf{G}_0 \\ \\ \end{bmatrix} + [1\,1\,1\,1\,1\,0\,0\,0\,0\,0]$$

$$H_2 \quad \begin{array}{c} 1\,1\,1\,1\,1\,0\,1\,1\,1\,0\,1 \\ 0\,1\,1\,0\,1\,1\,1\,1\,0\,1\,1 \\ 1\,0\,1\,1\,0\,1\,0\,1\,0\,0\,1 \\ 0\,0\,1\,0\,0\,0\,0\,1\,1\,1\,1 \\ 1\,1\,0\,0\,0\,0\,1\,0\,0\,0\,1 \\ 0\,1\,0\,1\,0\,1\,1\,0\,1\,1\,1 \\ 1\,0\,0\,0\,1\,1\,0\,0\,1\,0\,1 \\ 0\,0\,0\,1\,1\,0\,0\,0\,0\,1\,1 \end{array} \qquad \begin{bmatrix} \mathbf{k}_2 \end{bmatrix} \left[\begin{array}{c:c} & 0 \\ \mathbf{G}_0 & \vdots \\ & 0 \end{array}\right] + [1\,1\,1\,1\,1\,0\,1\,1\,1\,0\,1]$$

$$H_3 \quad \begin{array}{c} 1\,1\,1\,0\,0\,1\,0\,0\,0\,0\,0\,1 \\ 0\,1\,1\,1\,0\,0\,0\,0\,1\,1\,0\,1 \\ 1\,0\,1\,0\,1\,0\,1\,0\,1\,0\,0\,1 \\ 0\,0\,1\,1\,1\,1\,1\,0\,0\,1\,0\,1 \\ 1\,1\,0\,1\,1\,1\,0\,1\,1\,0\,0\,1 \\ 0\,1\,0\,0\,1\,0\,0\,1\,0\,1\,0\,1 \\ 1\,0\,0\,1\,0\,0\,1\,1\,0\,0\,0\,1 \\ 0\,0\,0\,0\,0\,1\,1\,1\,1\,1\,0\,1 \\ 1\,1\,1\,0\,0\,1\,1\,1\,1\,0\,1\,0 \\ 0\,1\,1\,1\,0\,0\,1\,1\,0\,1\,1\,0 \\ 1\,0\,1\,0\,1\,0\,0\,1\,0\,0\,1\,0 \\ 0\,0\,1\,1\,1\,1\,0\,1\,1\,1\,1\,0 \\ 1\,1\,0\,1\,1\,1\,1\,0\,0\,0\,1\,0 \\ 0\,1\,0\,0\,1\,0\,1\,0\,1\,1\,1\,0 \\ 1\,0\,0\,1\,0\,0\,0\,0\,1\,0\,1\,0 \\ 0\,0\,0\,0\,0\,1\,0\,0\,0\,1\,1\,0 \end{array} \qquad \begin{bmatrix} \mathbf{k}_3 \end{bmatrix} \left[\begin{array}{c:c} & 0\,0 \\ \mathbf{G}_0 & \vdots\;\vdots \\ & 0\,0 \\ \hdashline 0\,0\,0\,0\;\,0\;\,0\,1\,1\,1\,0 & 1\,1 \end{array}\right] + [1\,1\,1\,0\,0\,1\,0\,0\,0\,0\,0\,1]$$

$$\mathbf{G}_0 = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}$$

Figure 6.5: VLEC code C_{14} (8@10,5; 8@11,5; 16@12,5; 3,2) and its horizontal linear sub-codes

10@12,5; 3,2) with free distance five and average codeword length of 10.09 bits. It is also worthwhile to note that the design d_{free} is not always reached (due to the minimum converging distance) and it may be necessary to iterate the construction by increasing the design value for $d_{\min}$ to achieve the required d_{free}. With each iteration the average codeword length will be increased.

6.4. Heuristic Construction Algorithm

In order to solve the problems with the code-anticode construction, a heuristic construction algorithm was devised, which uses a computer search. The aim of this

algorithm is to construct a VLEC code with specified overall minimum block, diverging and converging distances (and hence a minimum value for d_{free}) and with codeword lengths matched to the source statistics so as to give a minimum average codeword length for the specified free distance and the specified source.

We will now describe the basic construction concept. First, a minimum codeword length, L_1, must be specified. This must at least be greater than or equal to the minimum diverging distance required. Then, a fixed-length code with this length and with minimum distance equal to $b_{\min}$ with a maximum number of codewords must be constructed. For a horizontally linear VLEC code, all fixed-length codes constructed with this algorithm must be linear or a coset. Since this construction uses a computer search, then this fixed-length code is also constructed using a computer search. One such search technique is the *greedy algorithm* [Pless, 1992] which will be given in Section 6.4.1. Let the code so constructed be C and let the number of codewords in C be s_1. Next, all the possible L_1-tuples which are at distance $d_{\min}$ from the codewords of C are listed. Let this set of words be W. Obviously, if $d_{\min} \geq b_{\min}$, then W will be empty. However, the bound on d_{free} given by expression (3.18) suggests that it is best to choose $b_{\min} = d_{\min} + c_{\min}$. Therefore we may take $d_{\min} = \lceil d_{\text{free}}/2 \rceil$ and $b_{\min} = d_{\text{free}}$. In which case, $d_{\min} < b_{\min}$ and such words are possible to find. So for the moment we are going to assume that W is not empty. Note that the minimum distance of the words in W is not specified for now. Next, the number of words in W is doubled by increasing the words' length by one bit by affixing first a '0' and then a '1' to the rightmost position of all words in W. So now W contains words of length L_1+1. These words are then checked with the codewords in C. Those words in W which satisfy the minimum converging distance required are retained, the others are discarded. So at the end of this operation, we are left with a set of words which when compared to the codewords of C satisfy the required minimum diverging and converging distances. The only other requirement left to satisfy is that these words, being of the same length, must have minimum distance at least equal to $b_{\min}$. So here again we must choose the maximum number of words (s_2) from what is left in W such that these words form a fixed-length code with minimum distance at least $b_{\min}$. Algorithms to achieve this are given in Sections 6.4.1

and 6.4.2 for horizontally linear and non-linear VLEC codes respectively. These words are then added to the codewords already in C to form a VLEC code ($s_1@L_1$, $b_{\min}$; $s_2@(L_1+1)$, $b_{\min}$; $d_{\min}$, $c_{\min}$). This whole procedure is then repeated by next considering all (L_1+1)-tuples which satisfy the minimum diverging distance to all the codewords in C and are also at minimum distance $d_{\min}$ to those codewords in C of the same length, then affixing the extra bit to these words, extracting those words which satisfy the minimum converging distance and finally choosing the maximum number of words which satisfy the minimum block distance. This algorithm stops either when there are no further possible words to be found, or else when the required number of codewords is reached.

The problem with this basic construction algorithm is that if L_1 is chosen to be too small, not enough codewords may be found, whereas if L_1 is chosen to be too large, the average codeword length would be non-optimal. In addition, the codeword lengths are not matched to the source statistics. In order to have more control of the code construction, we must alter the basic algorithm in two important aspects. First, when finding the maximum number of words which satisfy the minimum block, diverging and converging distances with some given length, we must allow the possibility of eventually dropping some of these words. This may enable us to find more codewords of longer length than otherwise would be possible, hence increasing the possibility of finding the required number of codewords. The second alteration that is required is that some codeword lengths may be allowed to be skipped, i.e. we may allow the affixing of more than one bit at a time to the set W of words which satisfy the diverging distance.

Since this algorithm is general enough also to construct non-linear VLEC codes, we will first describe the general algorithm for any VLEC code and then point out the changes that must be incorporated in order to ensure that the VLEC code so constructed is horizontally linear.

Figure 6.6 gives the flowchart for the full algorithm for constructing good VLEC codes using the basic heuristic construction algorithm highlighted above. This algorithm incorporates the necessary alterations discussed above in order to find the necessary number of codewords. So when it is no longer possible to find more words with the given

minimum diverging distance (i.e. no more codewords of longer length are possible), then one or more codewords from the last group of codewords with the longest length in C is deleted. How this is done is discussed in Section 6.4.3. But by performing this operation, more (longer) words may now be found which satisfy the minimum diverging distance. When there is only one codeword left in the last group of codewords with the longest length, then this is deleted from the code. The set of words W with length equal to the longest codeword length in C satisfying the minimum diverging distance is again derived, but this time more bits are affixed to these words. So if the previous time only one bit was affixed, now two bits will be affixed. This will increase the number of words in W by four times and hence more codewords satisfying the required conditions may now be found. The details of this algorithm are shown in Figure 6.6.

These alterations to the basic algorithm will also make it possible to produce more than one VLEC code with the specified parameters and number of codewords, each one with a different codeword length distribution. In addition, the whole algorithm given by Figure 6.6 may be repeated by incrementing the starting codeword length L_1 to give yet more VLEC codes. In order to limit this search, some maximum codeword length is also specified. The generation of this possible set of VLEC codes will enable us to match the codeword lengths to the source statistics in a rather brute force way by calculating the average codeword length for each code constructed and choosing that one which gives the minimum average codeword length for the given source. This procedure can be considerably sped up by incorporating the code selection process in the same construction algorithm, because for most of the codes constructed with this algorithm, the average codeword length becomes larger than for codes found earlier (or some specified value) even after the first few codewords are found, in which case the construction for that particular code is stopped and the next one considered immediately.

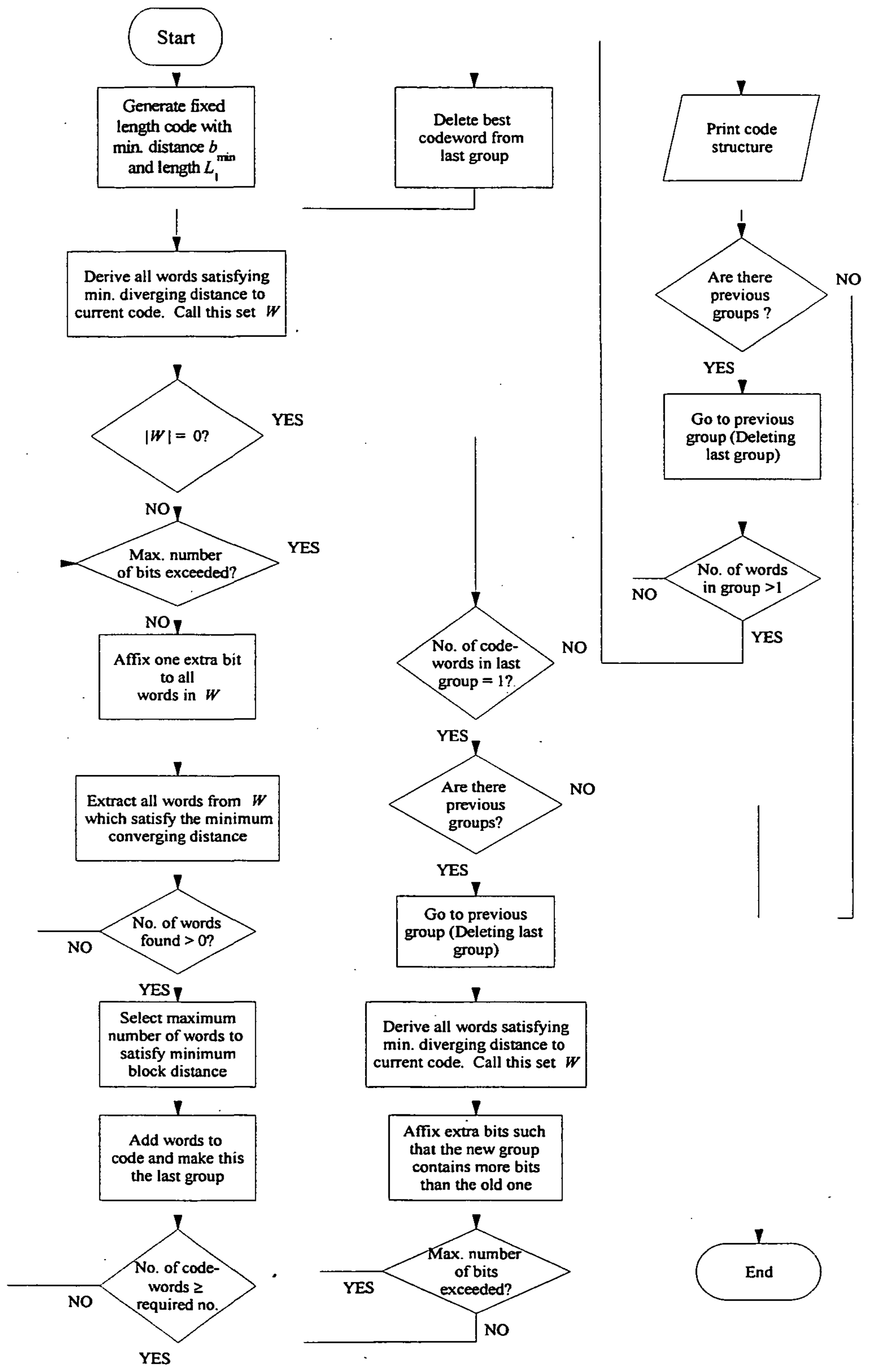

Figure 6.6: Heuristic construction algorithm for VLEC codes

6.4.1. Choosing a Good Fixed-Length Coset Code from a Given Set of Words

The greedy construction algorithm for a fixed-length block code with length n and minimum distance d is as follows. List all 2^n n-tuple words in lexicographic order. The first codeword chosen is the all zero codeword. Let the code thus formed be C.[2] The next codeword to be put in C is the next one in the list of n-tuples which is at a distance at least d from all the other codewords in C. This procedure is repeated for all possible n-tuples. It is very easy to see that the code so constructed has minimum distance d. Pless [1992] has also proved that this code is linear. We shall use this algorithm later in Section 6.4.2 to construct non-linear VLEC codes. However, since in the construction algorithm given in Section 6.4 the codewords must be chosen from a fixed set of words which in general does not contain all possible 2^n n-tuples and since this will affect the linearity of the code, then we will use the following algorithm to construct a linear code or a coset. We shall call this the *search and add algorithm*. Note that we will still assume that the words in the allowed search space are in lexicographic order.

1. Pick up the first word from the allowed set as the first codeword. This will also be the coset leader.
2. Go down the ordered set and find the first word at distance d or more from the previously chosen set of codewords. If one is not found, then stop; else proceed to the next step.
3. Add (modulo-2) this word to each previously selected codeword together with the coset leader (i.e., three words are added together in each case). If all the additions result in words in the allowed set, then choose all the words resulting from these additions as codewords, otherwise discard these words.
4. Repeat from Step 2 until there are no other words satisfying the required conditions.

Theorem 6.3: The code constructed using the search and add algorithm is a coset of a linear code with minimum distance d.

[2] So initially C has just one codeword.

Proof: The proof is by induction. Obviously, when choosing the first two codewords, the code is trivially a coset with minimum distance d. Let $\mathcal{F}$ be an intermediate code so constructed, and assume that it is a coset. Now Step 2 ensures that the next chosen word c is at Hamming distance d from all codewords in $\mathcal{F}$. Since the remaining words are generated through the addition of c to all codewords in $\mathcal{F}$ together with the coset leader, then obviously the resultant set of words is also a coset (with the same coset leader). Let $a, b \in \mathcal{F}$, where a is the coset leader and b some other codeword not equal to a. Then $H(a, b) \geq d$. Also, $H(a, c) \geq d$ and $H(b, c) \geq d$ as required by Step 2. Now the set of words generated by Step 3 are of the form $a + b + c = d$. But, $H(d, a) = H(a + b + c, a) = W(a + b + c + a) = W(b + c) = H(b, c) \geq d$. Similarly, $H(d, b) \geq d$. Hence the resultant code is a coset with minimum distance d. ■

6.4.2. Choosing a Good Fixed-Length (Non-Linear) Code from a Given Set of Words

The greedy algorithm may be used directly to construct a fixed-length code of length n from a given set of words which may not necessarily contain all possible n-tuples. In this case we will call this the *restricted greedy algorithm*. In fact, the code generated with this algorithm is, in general, non-linear. The advantage of this algorithm is that it is easy to implement and is very fast. The disadvantage is that the code produced is not always very good.

A better algorithm is the following, which we shall refer to as the *majority voting algorithm*. This selects a good fixed-length code, with minimum distance d, from a given set of words W.

1. Initialise the required code C to the null code (i.e. no codewords selected).
2. For each word in W determine the number of other words in W which are at least distance d from the given word.
3. Choose that word which has got the maximum number of other words which satisfy the minimum distance requirement. If there is more than one such word, then choose an arbitrary one. Put this word in the required code C.

4. Remove from W all those words which do not satisfy the minimum distance requirement to the chosen codeword.

5. Repeat Steps 2-4 with the new W. Each time the selected codeword is added to the code C. The algorithm ends when W is empty.

The main disadvantage with the above algorithm is that as the number of words in W increases it becomes too computationally expensive and the restricted greedy algorithm will then be preferred. Table 6.1 gives a comparison between the number of codewords generated by the greedy algorithm (GA) and the majority voting algorithm (MVA) together with the known maximum number of codewords possible [MacWilliams & Sloane, 1978] for various values of n and d. The allowed set of words in this case was all possible n-tuples. Note that except for the codes produced with $n = 12$ and 13 with $d = 3$, the MVA finds more codewords than the GA. The performance of the MVA is even better when compared to the restricted greedy algorithm when the set W does not contain all possible n-tuples.

n	$d = 3$			$d = 5$			$d = 7$		
	GA	MVA	Best	GA	MVA	Best	GA	MVA	Best
5	4	4	4	2	2	2			
6	8	8	8	2	2	2			
7	16	16	16	2	2	2	2	2	2
8	16	16	20	4	4	4	2	2	2
9	32	36	40	4	6	6	2	2	2
10	64	72	80	8	12	12	2	2	2
11	128	129	160	16	24	24	4	4	4
12	256	196	256	16	25	32	4	4	4
13	512	356	512	32	48	64	8	8	8

Table 6.1: Comparing the number of codewords found using the greedy algorithm and the majority voting algorithm when W contains all possible n-tuples

6.4.3. Deleting a Codeword

We have seen that in order to increase the number of codewords possible in a given code, some shorter length codewords must be deleted from the VLEC code such that more longer length codewords may be found. The problem is which codeword or group of

codewords is best deleted so as to maximise the number of possible new (longer) codewords.

In the case when the required code needs to be horizontally linear, then we cannot delete one codeword at a time, unless the codeword to be deleted happens to be in a sub-code consisting of just two codewords, otherwise we would destroy the linearity. In this case no optimisation was sought, instead the method adopted was to delete the lower half of the respective sub-code. Since the codewords are in lexicographic order, then this would result in another linear code with half the number of codewords as in the original one.

For the case of non-linear VLEC codes, some sort of optimisation was sought in order to maximise the number of codewords possible in a code. Hence when deleting a codeword from a code, the one deleted is that which will result in the greatest number of new words satisfying the minimum diverging distance. This is found through an exhaustive search. This of course is far from optimum but was found to give the best compromise between the computation speed and the resultant code optimality. There are in fact two main problems with this approach. First, the possible number of words that may be found satisfying the minimum diverging distance to the remaining codewords is maximised for each codeword deleted, in turn. However, this number could be further maximised if the deleted codewords are considered simultaneously. The second problem is that by deleting that codeword which gives the maximum number of words which satisfy the minimum diverging distance, it does not necessarily maximise the number of words which will also satisfy the minimum converging and block distances, even though in this case there would be more words to choose from.

6.5. Comparing Constructions

In order to assess better the qualities of the various construction algorithms presented in the earlier sections, we compare in Table 6.2 the codes produced by each respective algorithm for the 26-symbol English source given in Table A.1, for various values of d_{free}. Note from this table that the codes with the shortest average codeword length are those

produced with the heuristic construction making use of the majority voting algorithm to choose codewords of the same length satisfying the minimum block distance. This is somewhat as expected, since the heuristic algorithm in general matches the codeword lengths to the source statistics, and the majority voting algorithm uses the best algorithm to choose codewords of the same length with a specified minimum distance.

Also, note from Table 6.2 that the horizontally linear codes constructed with the heuristic algorithm are almost as good as the non-linear codes. However, on closer examination we observe that the horizontally linear codes so produced have horizontal sub-codes containing only few codewords. This of course offsets somewhat any gain obtained in having horizontal linearity. If we were to restrict the number of horizontal codes produced, and hence increasing the number of codewords per sub-code, this will result in increased average codeword length, as shown in Table 6.3. It is interesting to note that one of the codes produced in this way has the same parameters as C_{14} generated with the code-anticode construction in Example 6.1.

Another important aspect of the heuristic construction algorithm is the design value for $d_{\min}$. It was stated that the best value for $d_{\min}$ is $\lceil d_{\text{free}}/2 \rceil$. In fact, we could also choose $d_{\min}$ to be $\lfloor d_{\text{free}}/2 \rfloor$ although in this case, for odd d_{free}, it will result in a slightly slower construction, since more words will be found satisfying $d_{\min}$ in this case, and hence more computations will be required. The reason for choosing this value of $d_{\min}$ is so that $d_{\min}$ and $c_{\min}$ are nearly equal.[3] If $d_{\min}$ is greater than $\lceil d_{\text{free}}/2 \rceil$, then less words will be found satisfying this value for $d_{\min}$. Even though the corresponding value of $c_{\min}$ required will be smaller, the net result is that of increasing the average codeword length. Similarly, if $d_{\min}$ is chosen to be smaller than $\lfloor d_{\text{free}}/2 \rfloor$, the required minimum converging distance will correspondingly increase and this will also reduce the number of words satisfying $c_{\min}$. This is shown in Table 6.4, where different values for $d_{\min}$ were used to construct VLEC codes with $d_{\text{free}} = 5$ for the 26-symbol English source.

[3] Note that $d_{\min} + c_{\min} = d_{\text{free}}$.

Method	Average length	Code
$d_{\text{free}} = 3$		
α-prompt construction (Section 2.6.2)	7.830	(15@7,3; 11@14,3; 3,0)
Code-anticode construction	8.096	(16@8,3; 16@9,3; 2,1)
Heuristic construction (Horizontally linear)	6.741	(1@4,-; 4@5,3; 1@6,-; 2@7,5; 4@8,3; 2@9,6; 2@10,6; 4@11,3; 2@12,3; 4@13,3; 2,1)
Heuristic construction (Restricted greedy algorithm)	6.475	(2@5,4; 4@6,4; 7@7,3; 6@8,3; 6@9,3; 1@10,-; 2,1)
Heuristic construction (Majority voting algorithm)	6.370	(1@4,-; 1@5,-; 5@6,3; 6@7,3; 3@8,3; 4@9,3; 4@10,3; 1@12,-; 1@13,-; 2,1)
$d_{\text{free}} = 5$		
α-prompt construction (Section 2.6.2)	11.03	(23@11,5; 3@19,5, 5,0)
Code-anticode construction	10.46	(8@10,5; 8@11,5; 10@12,5; 3,2)
Heuristic construction (Horizontally linear)	8.729	(1@6,-; 1@7,-; 4@8,5; 4@9,5; 4@10,5; 2@11,5; 2@12,8; 2@13,10; 2@14,7; 2@15,6; 2@16,6; 3,2)
Heuristic construction (Restricted greedy algorithm)	8.690	(1@6,-; 2@7,5; 2@8,5; 4@9,5; 4@10,5; 4@11,5; 4@12,5; 4@13,5; 1@14,-; 3,2)
Heuristic construction (Majority voting algorithm)	8.467	(1@6,-; 1@7,-; 4@8,5; 5@9,5; 5@10,5; 6@11,5; 4@12,5; 3,2)
$d_{\text{free}} = 7$		
Heuristic construction (Horizontally linear)	10.85	(1@7,-; 1@8,-; 2@9,7; 2@10,8; 2@11,8; 2@12,10; 2@13,8; 1@14,-; 4@15,7; 4@16,7; 2@17,8; 3@18,7; 4,3)
Heuristic construction (Restricted greedy algorithm)	11.03	(1@8,-; 1@9,-; 2@10,7; 3@11,7; 4@12,7; 5@13,7; 6@14,7; 4@15,7; 4,3)
Heuristic construction (Majority voting algorithm)	10.70	(1@7,-; 1@8,-; 1@9,-; 2@10,7; 2@11,8; 4@12,7; 4@13,7; 5@14,7; 6@15,7; 4,3)

Table 6.2: Various codes for the 26-symbol English source constructed using different algorithms

Average Length	Number of sub-codes (σ)	Code parameters
8.729	11	(1@6,-; 1@7,-; 4@8,5; 4@9,5; 4@10,5; 2@11,5; 2@12,8; 2@13,10; 2@14,7; 2@15,6, 2@16,6; 3,2)
8.988	8	(2@7,5; 4@8,5; 4@9,5; 2@10,8; 2@11,6; 2@12,5; 4@13,6; 6@14,5; 3,2)
9.842	4	(4@9,5; 8@10,5; 8@11,5; 6@12,5; 3,2)
10.46	3	(8@10,5; 8@11,5; 10@12,5; 3,2)

Table 6.3: Horizontally linear VLEC codes for the 26-symbol English source with $d_{\text{free}} = 5$ with various numbers of sub-codes

Design $d_{\min}$	Average codeword length	Code parameters
5	11.02	(22@11,5; 4@16,5; 5,0)
4	10.06	(2@9,6; 8@10,5; 11@11,5; 1@12,-; 2@13,5; 2@14,6; 4,1)
3	8.467	(1@6,-; 1@7,-; 4@8,5; 5@9,5; 5@10,5; 6@11,5; 4@12,5; 3,2)
2	8.463	(1@6,-; 2@7,5; 4@8,5; 3@9,5; 4@10,5; 4@11,5; 5@12,5; 2@13,5; 1@14,-; 2,3)
1	10.02	(2@9,6; 9@10,5; 10@11,5; 1@12,-; 2@13,6; 2@14,6; 1,4)
0	11.02	(22@11,5; 4@16,5; 1,5)

Table 6.4: Variation of average codeword length with $d_{\min}$ for the heuristic construction

Table 6.4 also serves to show the non-optimality of this construction algorithm. Consider a VLEC code C (s_1@L_1, b_1; s_2@L_2, b_2; $\cdots$; s_σ@L_σ, b_σ; $d_{\min}$, $c_{\min}$) and let $\mathbf{c} = c_1c_2\cdots c_l$, $\mathbf{c} \in C$. Define the *swap operation on codeword* $\mathbf{c}$ ($\bar{\ }$) to be

$$\bar{\mathbf{c}} = c_l c_{l-1} \cdots c_1 . \tag{6.1}$$

Then the swap code of C, $\bar{C}$, is the code $\{\bar{\mathbf{c}} : \mathbf{c} \in C\}$. It is obvious that code $\bar{C}$ has the same performance as C with maximum likelihood decoding. In particular, the free distance of both codes is the same. In fact the only difference between the two codes is the values of the minimum diverging and converging distances, which are simply swapped, i.e. $\bar{C}$ is a (s_1@L_1, b_1; s_2@L_2, b_2; $\cdots$; s_σ@L_σ, b_σ; $c_{\min}$, $d_{\min}$) VLEC code. Hence optimum VLEC codes having swapped minimum diverging and converging distances must have the same average codeword length. From Table 6.4 it is clear that in this case the construction

algorithm does not find codes with the same average codeword length when $d_{\min}$ and $c_{\min}$ are swapped and hence is definitely (in general) sub-optimum.

6.5.1. Two-Length Error-Correcting Codes and VLEC Codes

The α-prompt construction with $d_{\text{free}} = 3$ given in Table 6.2 is in fact a two-length error-correcting code as considered by Dunscombe [1988]. This is constructed from the (7,4) Hamming base code. It is interesting to note that because this base code is perfect, the performance of this two-length code is practically the same both with the prefix decoding algorithm and with maximum likelihood decoding. However, due to the restrictions on the possible codeword lengths, the code rate for this code is low when compared to the best VLEC code constructed with the heuristic algorithm. In fact, for a slightly lower code rate, we can even construct a free distance five VLEC code. The performance of these codes is shown in Figure 6.7, together with their respective code rates. Their relatively poor performance is the main disadvantage with two-length error-correcting codes over the AWGN channel. In addition, as discussed in Chapter 5, these codes also suffer from the same disadvantages as fixed-length codes over channels which admit symbol deletion and/or insertion errors.

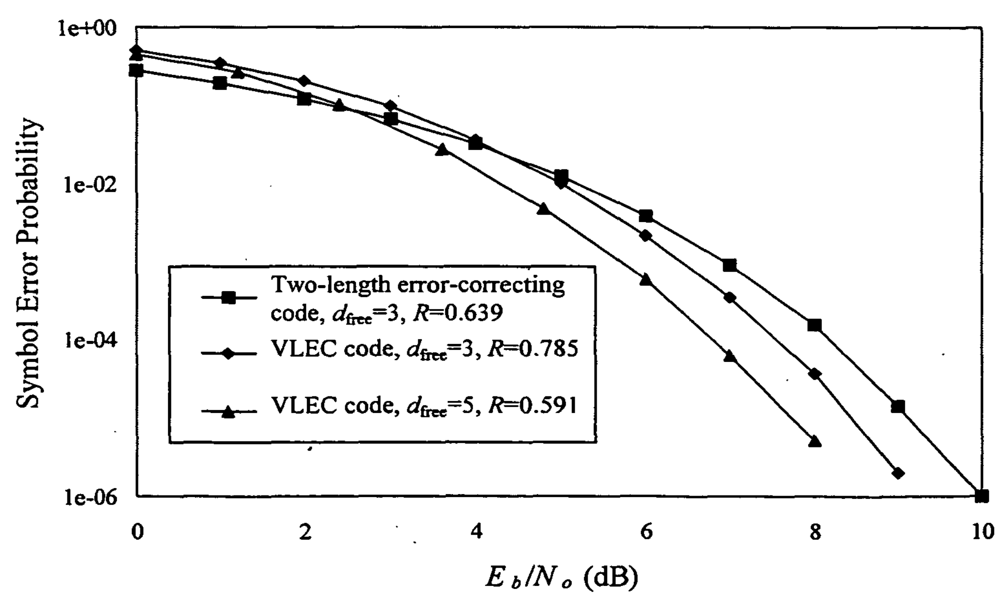

Figure 6.7: Comparing the performance of a two-length error-correcting code with VLEC codes

6.6. Comparing Performance of VLEC Codes with Standard Coding Techniques

Having discussed how to construct good VLEC codes, we now compare the performance of VLEC codes (with maximum likelihood decoding) to that of standard block and convolutional error-correcting codes. Also, since VLEC codes perform combined source and channel coding, then we will also compare their performance with standard cascaded techniques using separate source and error-correction coding.

Figure 6.8 shows the performance of the VLEC code C_{20} (1@6,-; 1@7,-; 4@8,5; 5@9,5; 5@10,5; 6@11,5; 4@12,5; 3,2) given in Table A.3, with free distance five and an average codeword length of 8.47 bits when used to encode the 26-symbol English source. This is compared both with BCH error-correcting codes and a convolutional code of the same minimum or free distance as appropriate. These codes were chosen so as to have approximately equal code rates. Hence, where appropriate, a source code (a Huffman code) was used to pre-encode the source before encoding with the error-correcting code. The Huffman code was chosen since it is also a variable-length code. The Huffman code for the 26-symbol English source has an average codeword length of 4.21 bits and hence a

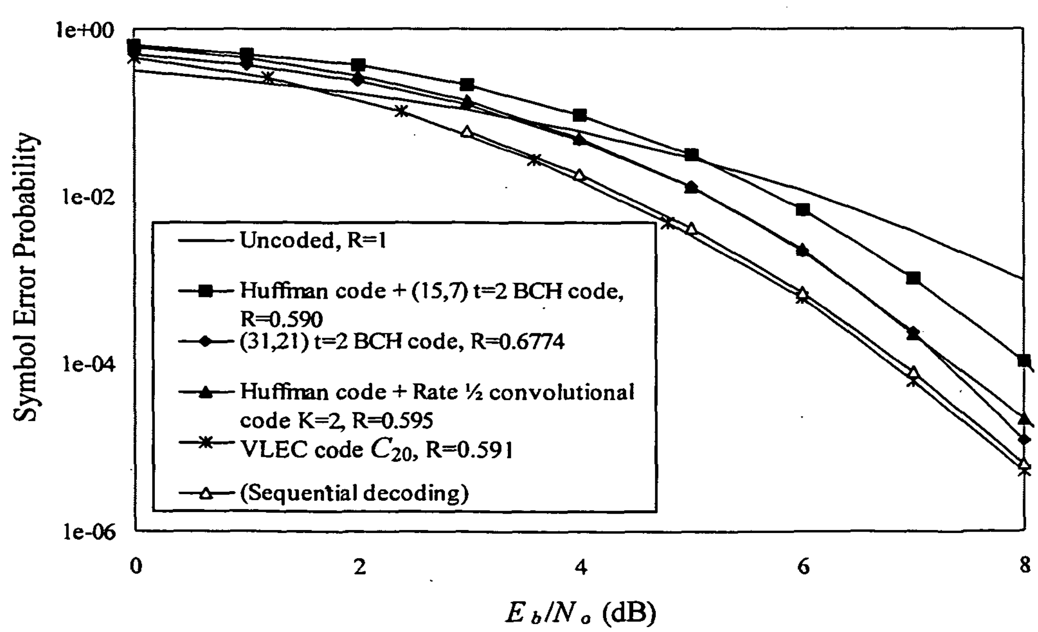

Figure 6.8: Free/Minimum distance 5 codes used to encode the 26-symbol English source

code rate of 1.19. In the results presented here both the BCH codes and the convolutional codes are decoded using hard decision, maximum likelihood decoding. For the BCH codes, (n, k) denotes a code with block length n and k information bits while t gives the number of correctable bit errors in a codeword, although, since maximum likelihood decoding is used, more than this number of errors may be corrected for some codewords. For the convolutional codes, K gives the constraint length[4]. The generator sequences for the convolutional codes used here, which give the maximal value for d_{free} for the given rate and constraint length, are taken from Table 11.1 of Lin and Costello [1983] . Note that the average codeword length of C_{20} is much shorter than the codeword length of either of the two BCH codes considered in Figure 6.8. For the case where no Huffman code is used, the source is simply encoded using 5 information bits for each symbol. Note that the VLEC code achieves a coding gain of slightly more than 0.5dB at a SEP of 10^{-4} over the best of the standard schemes, that using a cascaded Huffman/convolutional system, although the (31,21) BCH code starts to get the upper hand at high SNR due to its higher code rate.

Unfortunately, this modest coding gain is achieved at the expense of a large increase in the decoder complexity. Since with maximum likelihood decoding both VLEC and convolutional codes use practically the same algorithm (the Viterbi decoding algorithm), then it is easier to compare the complexity between these two schemes than between VLEC codes and block codes. Using similar reasoning as in Section 3.7, the number of comparisons and additions per source symbol required in the case of a rate k/n convolutional code with constraint length K are respectively given by

$$\text{Number of comparisons per source symbol} = \frac{\nu\left(2^k - 1\right)2^K}{k} \tag{6.2}$$

$$\text{Number of additions per source symbol}^5 = \frac{\nu \cdot n \cdot 2^{k+K}}{k} \tag{6.3}$$

where ν is the (average) number of information bits required to encode a source symbol.

[4] Where 2^K gives the number of distinct states in the convolutional encoder.

[5] Here it is assumed that the branch metric is computed for each output bit. Since in convolutional codes n is usually small, this number may be reduced by a factor n by implementing a lookup table to determine the branch metric for the n bits together.

Using equations (3.47) and (3.48) for VLEC codes and equations (6.2) and (6.3) for convolutional codes we require, on average, 212 comparisons and 2151 additions per source symbol in the case of the VLEC code while we only require, on average, 16.8 comparisons and 67.2 additions per source symbol in the case of the convolutional code, where in this case, because of the use of the Huffman code, $\nu = 4.20$ bits per symbol. The complexity in decoding the Huffman code is ignored in this case, since this is negligible compared to the complexity of the Viterbi decoder.

Figure 6.9 shows similar comparisons between the VLEC code C_{21} (1@7,-; 1@8,-; 1@9,-; 2@10,7; 2@11,8; 4@12,7; 4@13,7; 5@14,7; 6@15,7; 4,3) given in Table A.3, with average codeword length of 10.7 bits when used to encode the 26-symbol English source, and standard error-correcting codes with free/minimum distance seven. The (15,5) BCH code is preceded with the Huffman code for the English source. Again, the VLEC code performs slightly better than the half rate convolutional code. In order to match the code rate as much as possible, the convolutional code in this case is used with the uncoded source, i.e. $\nu = 5$ bits per symbol. The average number of comparisons and additions per source symbol for the VLEC code C_{21} are respectively 268 and 3488, whereas the corresponding numbers for the convolutional code are 80 and 320. Hence the decoding complexity for the VLEC code is an order of magnitude larger than that for the convolutional code with the same free distance. Therefore a larger free distance convolutional code may be used for the same complexity and in this case the performance of the convolutional code will be better than that of the VLEC code.

Similarly, Figures 6.10 and 6.11 give comparisons for codes C_{22} (2@7,5; 3@8,5; 4@9,5; 5@10,5; 8@11,5; 8@12,5; 8@13,5; 9@14,5; 9@15,5; 10@16,5; 13@17,5; 11@18,5; 16@19,5; 17@20,5; 5@21,5; 3,2) with average codeword length of 9.60 bits and C_{23} (32@15,7; 32@16,7; 64@17,7; 5,2) with average codeword length of 15.1 bits, both given in Table A.4, when used to encode the 128-symbol ASCII source also given in this table. The source statistics for this source were derived from the source file of a *C* program. The Huffman code for this source has an average codeword length of 5.10 bits and a code rate of 1.38. Note that in Figure 6.10, the cascaded Huffman code with the

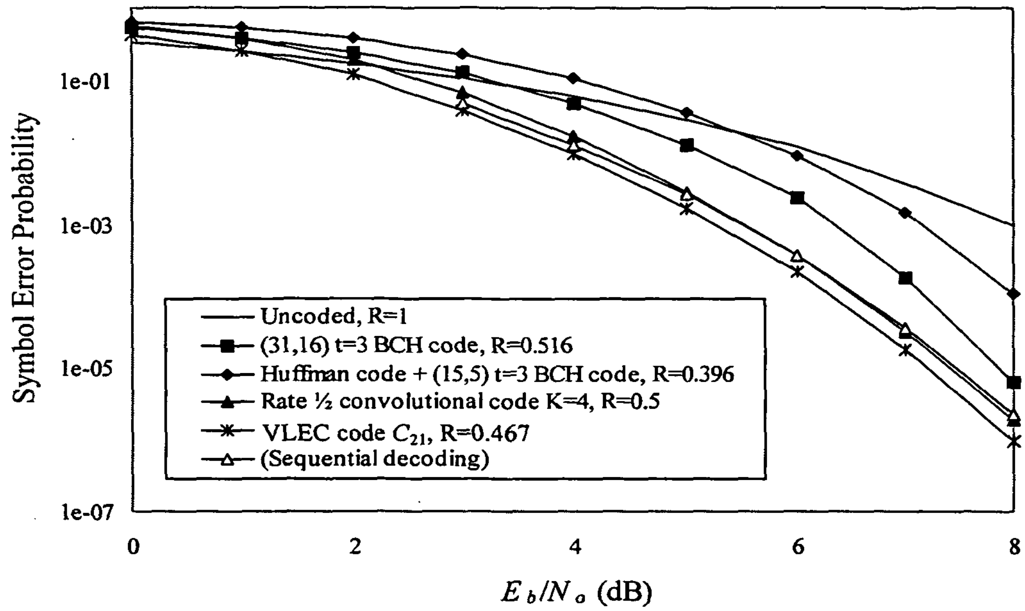

Figure 6.9: Free/Minimum distance 7 codes used to encode the 26-symbol English source

(31,21) BCH code performs better than the VLEC code at high SNR. This is mainly due to the fact that this cascaded scheme has a higher code rate.

Referring to Figure 6.10, the coding gain of C_{22} compared to the convolutional code at a SEP of 10^{-5} is about 1dB. If we were to compare the number of comparisons and additions necessary in this case, we find that the convolutional code requires on average 20.4 comparisons and 81.6 additions per symbol, where due to the Huffman code, ν=5.10 bits per symbol. Whereas to decode C_{22} we require 1219 comparisons and 19210 additions per symbol. Notice the large difference in the number of additions required. This is due to the fact that the branch labels in the trellis for the convolutional code contain only n bits (in this case 2), whereas the branch labels for the VLEC code contain the codewords themselves, in this case with a maximum codeword length of 18 bits. Also, the number of transitions going to a state is 2^k in the case of convolutional codes, while the equivalent number for VLEC codes is s.

For the codes in Figure 6.11, the convolutional code requires 112 comparisons and 448 additions per symbol, whereas the VLEC code C_{23} requires 1919 comparisons and 31429 additions per symbol. It is interesting to note that by increasing the free distance

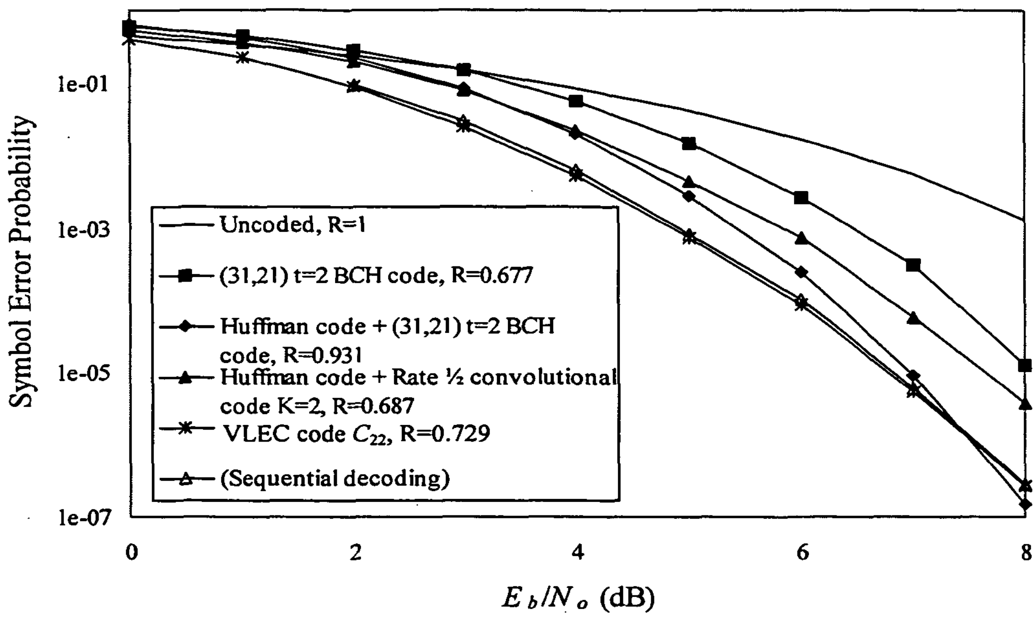

Figure 6.10: Free/Minimum distance 5 codes used to encode the 128-symbol ASCII source

from 5 to 7, the number of operations in the case of the convolutional code increases by a factor of five. This is somewhat inflated by the fact that for the free distance 7 code, the source is not encoded with the Huffman code, resulting in ν=7. If we allow for this fact, then the increase is a factor of 3.5. Whereas in the case of the VLEC codes the corresponding increase is a factor of 1.5.

The code rate for code C_{23} having d_{free} = 7 is not very good, because this code was constructed using the code-anticode construction and is hence a horizontally linear VLEC code. This means, however, that for this code we can greatly reduce the decoding complexity by decoding each sub-code independently using their linearity. If we ignore the complexity required for this step, then in this case, since there are three fixed-length sub-codes in C_{23}, the number of comparisons and additions per symbol are respectively 30 and 725 which are comparable to those required by the convolutional code. The number of additions may be further reduced to 15 if the decoder for the sub-codes supplies also the metric value for the decoded codewords.

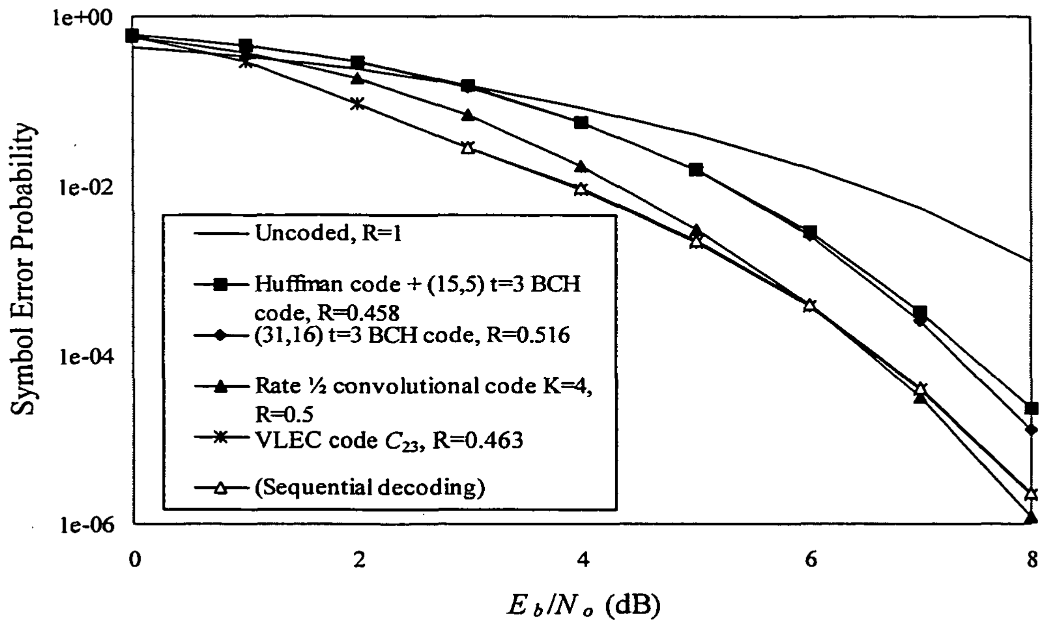

Figure 6.11: Free/Minimum distance 7 codes used to encode the 128-symbol ASCII source

The three sub-codes for C_{23} have parameters (15,5), (16,5) and (17,6). So comparing the decoding of this VLEC code to the BCH codes, we now require three decoders instead of just one. However these could be implemented in parallel to speed up the decoding. In the case of fixed-length block codes the decoding window moves n bits at a time, i.e. once a codeword is decoded, n bits from the received bit sequence are discarded and the next n bits are considered. In the case of maximum likelihood decoding of VLEC codes, this is not so. When the codeword lengths have no common factor, as in this case, the decoding window is moved only one bit at a time. Therefore the three sub-code decoders are used to decode codewords of lengths 15, 16 and 17, but then only one bit from the input bit sequence is discarded and the process repeated for the next decoding. Hence, since the average codeword length for C_{23} is 15.1 bits, on average we need to decode 3×15.1 codewords for each symbol transmitted, whereas for the BCH codes we only need to decode v/k codewords.

Also shown in Figures 6.8-6.11 is the performance of each corresponding VLEC code with the stack decoding algorithm given in Chapter 4, with a stack size of 50. Although all these codes, with the exception of C_{23}, are sequentially catastrophic, we note

that the performance is almost as good as that with maximum likelihood decoding. However, comparing the required number of comparisons and additions necessary for these two algorithms using equations (3.47), (3.48), (4.15) and (4.17), we deduce that, for high SNR, the number of comparisons in the sequential algorithm is reduced by a factor of $\frac{L_{average}(s-1)}{g(s-\sigma)}$ and the number of additions by a factor of $\frac{L_{average}}{g}$, since for high SNR $\xi \rightarrow 1$. In Table 6.5 we summarise the number of computations required for the codes considered here. Note that "MLD" and "Sequential" in this table give the number of computations required, for each corresponding VLEC code, with maximum likelihood and sequential decoding respectively, while L_{average} and σ give respectively their average codeword length and the number of different codeword lengths. We note that the number of computations required for sequential decoding the VLEC codes used to encode the 26-symbol source is now comparable to that required to decode the convolutional codes. However in the case of the VLEC codes used to encode the 128-symbol source, the number of additions required with sequential decoding is still an order of magnitude more than that required for the convolutional codes. Again, for code C_{23} this number can be further reduced, as discussed above, by taking advantage of the fact that this code is horizontally linear.

	26-symbol				128-symbol			
	$d_{\text{free}} = 5$		$d_{\text{free}} = 7$		$d_{\text{free}} = 5$		$d_{\text{free}} = 7$	
	Comparisons	Additions	Comparisons	Additions	Comparisons	Additions	Comparisons	Additions
Convolutional	16.8	67.2	80.0	320	20.4	81.6	112	448
MLD	212	2151	268	3488	1219	19210	1919	31429
Sequential	19	254	17	326	113	2001	125	2081
L_{average}	8.47		10.7		9.60		15.1	
σ	7		9		15		3	

Table 6.5: Comparing number of computations required for convolutional and VLEC codes

6.7. Conclusion

The two constructions for VLEC codes given in this chapter both involve, to some degree, a computer search. In the case of the code-anticode construction, a computer search is necessary in order to find a suitable modification vector (or a column permutation) so as to get the required minimum converging distance. The problem in designing a VLEC code with a specified minimum converging distance in the first place is difficult because in computing the converging distance not all codewords are shifted the same number of bits and the resultant structure will become non-linear even if the code is horizontally linear. Another disadvantage of this construction is that the resultant codeword lengths are not matched to the source statistics and consequently the resultant code will have non-optimal average codeword length for the given free distance.

The second algorithm given to construct VLEC codes relies completely on a computer search and uses some heuristics to limit the search space. This construction may be adapted to give both horizontally linear and non-linear VLEC codes. The codes so constructed have codeword lengths matched to the source statistics and hence have a shorter average codeword length than codes constructed with the code-anticode construction. The disadvantage of this algorithm is that it becomes impractical for sources with a large number of symbols. The maximum number of symbols deemed practical with current computer technology is about 128 symbols. The main problem is the maximum codeword length; as this increases, the search space increases exponentially.

No attempts were made to construct the horizontal sub-codes of horizontally linear VLEC codes as cosets of cyclic error-correcting codes. However we envisage no problems with this approach. This will further facilitate the encoding and decoding of VLEC codes.

The number of codewords in a horizontal sub-code of a horizontally linear VLEC code must be a power of two. However, for the last sub-code, i.e. the one with maximum codeword length, this condition may be relaxed by having a number of unused codewords. This is necessary so as to obtain the required overall number of codewords for the VLEC code. This does not present any problems in decoding such a code. The only alteration necessary is to assign a large cost to these unused codewords, so that if one of them is

chosen by the decoder of the corresponding sub-code (this is possible in the presence of noise, of course), then the modified Viterbi decoder would in any case disregard the corresponding path in the trellis.

A simulated annealing approach [El Gamal *et al.*, 1987] to the design of good VLEC codes was also attempted, without much success. Although the algorithm works well for the design of fixed-length codes[6], it breaks down when it comes to VLEC codes. Here, the energy function must incorporate not only the minimum distance but a host of other parameters, *viz.* minimum block, diverging and converging distances and the average codeword length. Also the fact that the codeword lengths are not equal makes it difficult to define a suitable energy function. For instance two short codewords may be only a specified distance from each other, but two long ones may be at a much larger distance more easily. Hence the algorithm tends to produce VLEC codes where the short codewords do not conform with the required parameters.

It is interesting to note that the construction of equivalent VLEC codes is not trivial, where equivalent VLEC codes are codes having an identical distance spectrum. In fact the only equivalent code construction known to us which may be applied in general is that obtained through swapping the codewords (see Section 6.5).

In the previous chapter we have seen that it is advantageous to construct non-catastrophic VLEC codes. Unfortunately both the constructions given in this chapter are not easily adaptable to prevent constructing non-catastrophic codes and sometimes hand tweaking would be necessary in order to avoid a catastrophic code. However, we do not consider catastrophic behaviour of VLEC codes of prime importance with regard to their performance.

Finally we have seen that although VLEC codes do achieve a slight coding gain over standard coding schemes of comparable coding rate and minimum/free distance, this is usually achieved at the expense of a large increase in the complexity of the decoder. For

---

[6] For instance, the maximum number of codewords found for a fixed-length code with block length 8 and minimum distance 3 using simulated annealing was 20, which is indeed the maximum number possible [MacWilliams & Sloane, 1978].

high SNR the decoding complexity may be drastically reduced in the case of VLEC codes by using sequential decoding. But obviously this is also a solution for convolutional codes, although for VLEC codes the size of the required stack is relatively much smaller. The number of comparisons and additions are not always useful in comparing the complexity between two decoding algorithms, although it is indicative. These numbers depend a lot on the actual implementation of the decoder. Equations (3.47) and (3.48) assume that the decoding is being achieved in software in a sequential manner. However, if the metric computation is done in hardware, for instance, then the number of required additions may be reduced a lot. For instance, if the maximum codeword length is L_{σ}, then two L_{σ}-bit registers together with an L_{σ}-bit binary adder may be used to evaluate the metric of any codeword in two clock cycles. In addition a number of these computations may be performed in parallel. Hence it will not be impossible to implement a VLEC maximum likelihood decoder in hardware operating at reasonable speeds.

Chapter 7.

Conclusion

In this thesis we have introduced a novel representation for VLEC codes which incorporates the spatial memory inherent in these codes due to their variable-length nature. This led to the derivation (in Chapter 3) of a maximum likelihood decoding algorithm based on the Viterbi algorithm. The performance of VLEC codes with this algorithm is much better than with the instantaneous algorithms found in the literature (see Chapters 2 and 3) and is also in fact slightly better than the performance of standard cascaded source and channel coding schemes employing Huffman codes (for source coding) and BCH or convolutional codes (for channel coding) with similar code rates and the same minimum or free distance (see Chapter 6). However, the main objective for combining source and channel coding is to obtain a reduction in the overall system complexity for the same performance. Unfortunately, we have shown in Chapter 6 that the complexity of maximum likelihood decoding for VLEC codes is much more than that for the separate schemes. The decoding complexity for VLEC codes at relatively high SNR may be reduced by using sequential decoding (Chapter 4). For VLEC codes with adequate CDF growth rate, the performance with sequential decoding is the same as with maximum likelihood decoding, but the decoding complexity is decreased by a factor of $L_{average}/g$ (for high SNR) and becomes, in some cases, comparable to that of the standard cascaded schemes.

The decoding complexity may be further reduced by using horizontally linear VLEC codes. The reduction factor is influenced by the number of horizontal sub-codes used; the less the number of sub-codes, the greater the reduction in the decoding complexity. However this must be balanced out against a possible decrease in the code rate due to non-

matching of codeword lengths to the source statistics, resulting in a longer average codeword length.

By using the modified Viterbi algorithm, soft-decision decoding for VLEC codes is very easy to implement and the decoding complexity is not increased much. The decoding complexity may be further reduced by performing trellis decoding for each of the horizontal fixed-length sub-codes[1] [Wolf, 1978] [McEliece, 1994]. It may even be possible to reduce the overall complexity by combining these two trellis structures, hence removing any redundancy in the number of states and/or transitions.

The main problem in using variable-length codes in general over the BSC is that of synchronisation, whereby simple substitution errors may cause a long error propagation because of loss of synchronisation. With VLEC codes the problem still exists; however, now some error patterns may be corrected and hence loss of synchronisation is avoided in these instances. By using maximum likelihood decoding this problem is reduced even further (at the expense of increased complexity) by checking every synchronisation position possible and choosing that message which gives the minimum number of bit errors[2]. This is done in the most efficient way possible by using either the Viterbi algorithm or, for high SNR, the stack algorithm. The instantaneous decoding strategies for VLEC codes treated in the literature (see Chapter 2) still fail miserably when they lose synchronisation. This problem with the instantaneous algorithms may be reduced by using two-length VLEC codes. However this is a half-way approach to combined source and channel coding in that it is not very effective in performing good source coding, since the codeword lengths cannot be matched well with the source statistics. We have also shown in Chapter 5 that these codes have practically the same disadvantages as fixed-length codes over channels which admit symbol deletions and insertions, whereas VLEC codes with codeword lengths having no common factor do not.

[1] This may be possible both for horizontally linear and non-linear sub-codes, although in the latter case it may be more difficult to implement.
[2] On the BSC this will also be the most likely one..

From the bounds obtained in Section 3.5.1 we have seen that the single most important parameter that affects the performance of VLEC codes over the BSC with maximum likelihood decoding is the free distance[3]. For most VLEC codes considered, the bound on d_{free} given by expression (3.18) was found to be met with equality. This is mainly due to the fact that in VLEC codes there always exist very short merged paths. If the code contains codewords of equal length, then the shortest merging occurs with paths with just one source symbol. In any case, there will always exist merged paths with just two source symbols. Hence, the three parameters b_{min}, d_{min}, and c_{min} are very indicative of the performance of a VLEC code, since in most cases the free distance will be directly determined by these parameters.

One direct consequence of the above is the interesting situation whereby the minimum block distance of the shortest length codewords can be the same as that for the longest length codewords, without affecting the performance of the code. From this fact we can deduce that there is a larger probability of correcting errors in the shorter codewords than in the longer ones. But this makes perfect sense also in light of the fact that the shorter length codewords are more probable (since they are mapped to the most probable source symbols). Hence, on average, the error correcting capabilities of a VLEC code is improved in this way. Therefore we can also claim that VLEC codes may offer unequal error protection to different source symbols, offering more protection to the more probable ones.

Using some of the newly defined properties of VLEC codes, we may characterise classical properties of variable-length codes for the noiseless case using new definitions. Hence, a code is non-singular iff $b_{min} > 0$. A code is uniquely decodable iff $d_{free} > 0$, since any two merging paths in the trellis would be distinguishable. A code has a finite decodable delay iff $d_{free} > 0$ and $d_c(\eta) > 0$ for some finite η, where $d_c(\eta)$ is the CDF for the VLEC code. A code is instantaneously decodable (i.e. it is a prefix code) iff $d_{free} > 0$ and $d_{min} > 0$. Finally, a code has a finite synchronisable delay iff the code is non-sequentially-

[3] This was also found to be the case, using simulation, for sequential decoding of non-sequentially catastrophic VLEC codes.

catastrophic and has $c_{\min} > 0$ (see Theorem 5.3)[4]. It is also conjectured that a code is exhaustive iff $d_{\text{free}} = 1$ and $A_1 = L_{\text{average}}$, where A_1 is the average number of converging pairs of paths at Hamming distance one and L_{average} is the average codeword length.

7.1. Scope for Further Research

Many open problems remain. Here we list some which we consider important.

Upper and lower bounds on the possible average codeword length for VLEC codes with a given free distance are required. These are not easy to derive due to the multitude of parameters involved. Because VLEC codes are used for combined source and channel coding, an optimum[5] VLEC code for one particular source may be sub-optimal for a different source. In addition, whereas for fixed-length codes the only parameter that needs to be bounded, for a given number of codewords and minimum distance, is the block length, in the case of VLEC codes the number of parameters is much more. These are the minimum and maximum codeword lengths, the number of different codeword lengths and the minimum block, diverging and converging distances required for the given free distance. These bounds, however, will enable us to judge how good a particular VLEC code is and may prompt further research to find a better construction for these codes. Furthermore, the heuristic construction algorithm, given in Section 6.4, which is not practical when constructing codes with a large number of codewords, has shown that the code-anticode construction is sub-optimal. Hence other construction techniques are required for large codes.

Due to the good synchronisation properties of VLEC codes found in Chapter 5 under symbol insertion or deletion errors, it is worthwhile to investigate the performance of these codes over channel models which admit these types of errors (such as for instance over fading channels) and compare their performance with standard coding schemes.

---

[4] The proof is slightly different, in that the decoder is not attempting to correct errors and hence does not need the GCD of the codeword lengths to be one, since the decoder does not need to assume that the first received bit correspond to a state in the trellis.

[5] Optimum in the sense that the average codeword length is the minimum possible for the given free distance.

It was found that the performance of sequential decoding depends on the growth rate of the CDF, using a rather simplistic model (see Section 4.4.2). This, however, has not been supported much by the simulation results obtained. A more accurate model must therefore be used. We also think that a different measurement replacing the CDF must be introduced, one which will also include the path probabilities in addition to their distance. For instance, a VLEC code may be sequentially catastrophic only for a single pair of paths, whose probability of occurrence is negligibly small. Obviously this will perform much better than another sequentially catastrophic code with many possible paths exhibiting catastrophicity, even though in this case the CDF for both codes may be identical. Bounds on the computational effort for sequential decoding will also be useful.

The relationship between the decoding window depth in the modified Viterbi decoder for a VLEC code and the constraint length of the code must be established theoretically. This relationship was briefly treated empirically in Section 3.6. This will enable the design of maximum likelihood decoders for VLEC codes with the minimum possible buffer size.

Of mathematical interest is the relationship between the forbidden states at the initial and final part of the trellis for a VLEC code and its codeword lengths. No simple relationship could be found. This could be of interest when implementing a sequential decoder, since these forbidden states must be eliminated at the end of the tree so that the correct number of bits will be decoded in the final path.

7.2. Some New Ideas

One of the problems with VLEC codes is that the free distance cannot be increased without increasing the average codeword length (for an optimum code), hence decreasing the code rate. This has a two-fold disadvantage for a constant information rate system. First, a larger bandwidth will be required, and secondly, due to the increased bit rate on the channel, the energy per bit will decrease (for a constant transmitter output power) resulting in more errors (i.e. the cross-over probability of the derived BSC will be higher). Both of these effects will increase the SEP and somewhat offset the coding gain achieved by

increasing the free distance. These problems may be solved using what we term as state-splitting VLEC codes. These are considered in Section 7.2.1.

A finite state machine representation for VLEC codes is also possible, and this may enable other mathematical techniques to be used to characterise the properties of VLEC codes with maximum likelihood decoding. This is treated in Section 7.2.2.

7.2.1. State-Splitting Variable-Length Error Correcting Codes

In order to increase the free distance of VLEC codes without increasing the average codeword length, we can employ a technique which we call *state-splitting*; hence the resultant codes will be known as *state-splitting VLEC (SSVLEC) codes*. The idea is similar to that in convolutional codes, where the longer the constraint length, the larger the free distance possible. A longer constraint length directly translates into more states in the trellis for the convolutional code and this permits longer unmerged paths and hence larger free distance. The technique we propose here is similar. The states in a VLEC code are related solely to the corresponding starting bit position of each codeword. In SSVLEC codes, these states are artificially split into other states. We define the *order* of a SSVLEC code to be the number of states representing a single position. Hence the VLEC codes considered in this thesis are SSVLEC codes of order one.

Consider code C_6 given in Table 3.3. This has constraint length 1.58, since we only require to store in memory three states to build all subsequent states in the trellis. However, if we were to split each one of these states in two, this will give a SSVLEC code of order two with constraint length 2.58. Hence, while the average codeword length remains the same, the constraint length increases. Now, however, each source symbol is no longer mapped to a single codeword, but to multiple codewords, dependent on the current state of the encoder. A possible code for this two symbol source is given in Figure 7.1. With this encoding, the free distance for the SSVLEC code is increased from four (for the original code C_6) to five, for the same code rate.

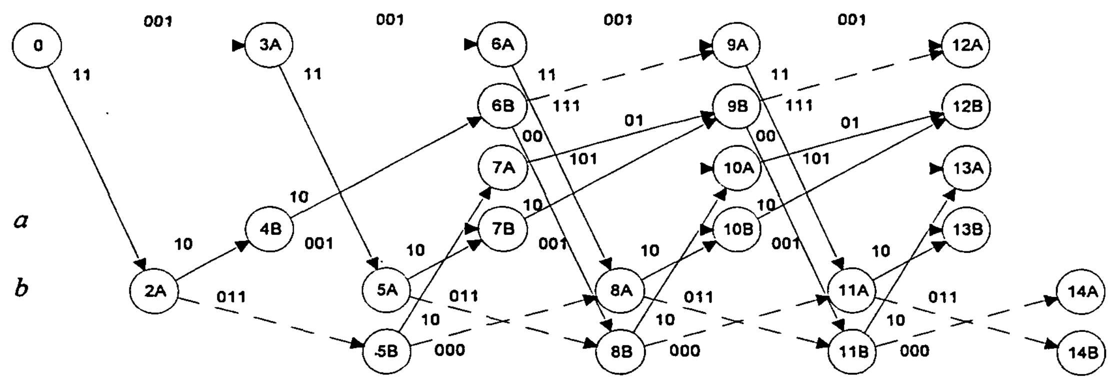

Figure 7.1: SSVLEC code of order two

Obviously now the complexity of the encoder and the decoder increases correspondingly. In addition, we must now devise other algorithms to construct good SSVLEC codes.

7.2.2. Finite State Variable-Length Error-Correcting Codes

The trellis constructed for a VLEC code C ($s_1@L_1$, b_1; $s_2@L_2$, b_2; $\cdots$; $s_\sigma@L_\sigma$, b_σ; $d_{\min}$, $c_{\min}$) in Chapter 3 has the disadvantage that the branch labels have a variable number of output bits, resulting in transitions to within the same stage of the trellis. Because of this, no finite state machine may be derived. We can eliminate this disadvantage by keeping the number of output bits per branch label fixed. We can choose this number to be any value, but the two most convenient ones are those corresponding to the minimum codeword length (L_1) and the maximum codeword length (L_σ). Obviously, to retain the same code mappings, we now need to vary the number of source symbols associated with each branch label.

Consider first when the output branch labels are taken to be of length L_1 bits. In this case, codewords of length greater than L_1 will have extra bits left over. These extra bits, forming a word $\mathbf{v}$, will determine the next state to which the encoder will jump. Each state will thus be associated with a unique word $\mathbf{v}$ which all branches emitting out of this state must output next. To be more precise, suppose that we are in some state $\mathbf{v}$, with $|\mathbf{v}| = l$, $l < L_1$, then the input source symbol a_i mapped to the codeword $\mathbf{c}_i = c_{i_1}c_{i_2}\cdots c_{i_{l_i}}$ will result in

a transition to state $c_{i_{L_1-l+1}} c_{i_{L_1-l+2}} \cdots c_{i_{l_l}}$ with output $\mathbf{v} c_{i_1} c_{i_2} \cdots c_{i_{L_1-l}}$.[6] The initial state is state λ, i.e. the state associated with the empty word (since initially there are no previous extra bits). Any state $\mathbf{v}$, with $\mathbf{v} = v_1 v_2 \cdots v_l$ and $l \geq L_1$ will have a "spontaneous" (i.e. with no input source symbols) transition to state $v_{L_1+1} v_{L_1+2} \cdots v_l$ with output $v_1 v_2 \cdots v_{L_1}$. All other states have a single transition to a next state (not necessarily different from the present one) for each one of the input source symbols. As an example, consider the state diagram for the simple VLEC code C_6, given in Table 3.3, shown in Figure 7.2, where ϕ indicates a "spontaneous" transition with no input symbols. It is very easy to see that because of the way the finite state machine is constructed, the output is the same as if the source has been encoded with code C using the mapping $a_i \rightarrow \mathbf{c}_i$. The only difficulty that may arise with this representation is at the end of the message. Here we must enforce that the encoder outputs also the word associated with the final state.

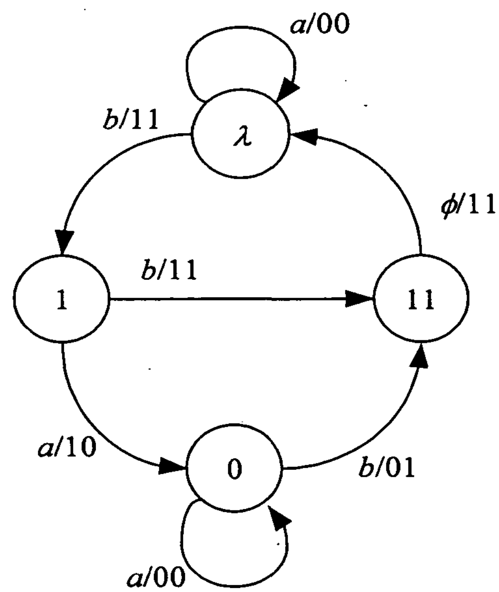

Figure 7.2: Finite state diagram for VLEC code C_6 with output branch labels of length L_1

The construction for the finite state machine for C taking L_σ as the number of output bits per branch label is very similar in principle, but results in an entirely different machine,

[6] i.e. the juxtaposition of the words $\mathbf{v}$ and $c_{i_1} c_{i_2} \cdots c_{i_{L_1-l}}$.

although the output for a given input message is the same[7]. One advantage with taking L_σ as the number of output bits is that we eliminate the "spontaneous" transitions. However now we may require more than a single input source symbol to effect a transition. The state diagram is constructed as follows. Again, the initial state is state λ associated with the empty word. If the encoder is in state $\mathbf{v}$, with $|\mathbf{v}| = l$, then the input source symbol sequence $\mathbf{a}_i = a_{i_1}a_{i_2}\cdots a_{i_m}$ which is mapped to the codeword sequence $\mathbf{c}_{i_1}\mathbf{c}_{i_2}\cdots\mathbf{c}_{i_m}$ causes the encoder to emit the output bits $\mathbf{v}\mathbf{c}_{i_1}\mathbf{c}_{i_2}\cdots\mathbf{c}_{i_{m-1}}\mathbf{p}$ and go to state $\mathbf{s}$, where $\mathbf{ps} = \mathbf{c}_{i_m}$ such that $|\mathbf{v}\mathbf{c}_{i_1}\mathbf{c}_{i_2}\cdots\mathbf{c}_{i_{m-1}}\mathbf{p}| = L_\sigma$, for some m. Again, we must enforce that the encoder outputs the word associated with the final state at the end of the message. Figure 7.3 gives the corresponding finite state diagram for code C_6.

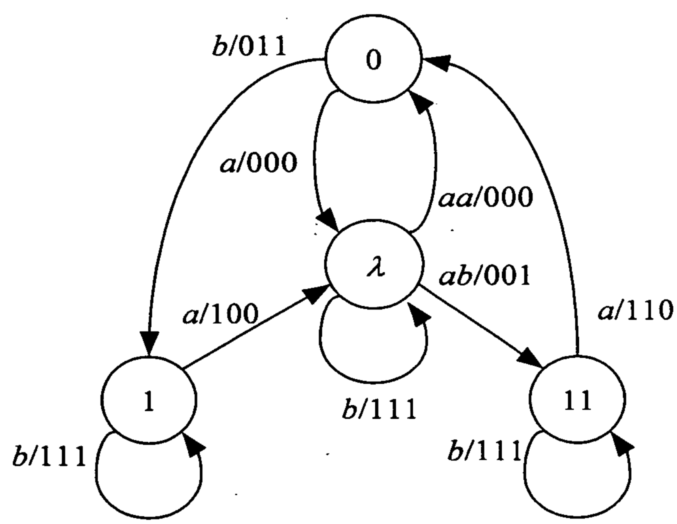

Figure 7.3: Finite state diagram for VLEC code C_6 with output branch labels of length L_σ

The implementation of maximum likelihood decoding for both these representations is straight forward using the Viterbi algorithm. In addition, these new representations may enable the use of other mathematical techniques with which to analyse VLEC codes. One interesting aspect of these representations is the number of states required in the finite state machine. For the particular case of code C_6, both the finite state machine representations

[7] It is the same if taken over the whole message; it will be different, however, if compared on a per source symbol basis.

(Figures 7.2 and 7.3) result in four states. However for other codes this number may be different. In addition, the number of states is also different from that required by the trellis representation presented in Chapter 3. For instance for code C_6 we only require three states in the trellis representation instead of the four required with the finite state machine representation.

One other interesting possibility arising from this representation that could be investigated is that of constructing a new class of codes, which we call *finite state VLEC (FSVLEC) codes*, where the "codeword lengths" are matched to the source statistics automatically. First, a finite state machine representation for a Huffman code for the given source is derived. The output branch labels are then replaced by new ones, of longer length, such that some required free distance is achieved. The requirement that the prefix of all output branch labels from state **v** is **v** may be dropped in this case to facilitate the design. The resultant code will then no longer be equivalent to a VLEC code. Some codes for an eight symbol source were designed using similar ideas to those used by Pollara *et al.* [1988] to construct finite state (fixed-length) codes. The resultant codes have good code rates and for a free distance three code, better performance than a VLEC code constructed with the heuristic algorithm with the same free distance.

FSVLEC codes are also considered by Piret [1982], who calls them comma-free error-correcting codes of variable length. The author only considers codes for the BMS and the finite state machine is constructed such that the output branch labels are only of length one bit. The codes so constructed are consequently of low rate resulting in poor performance.

References

Abramson N., *Information theory and coding*, New York, McGraw-Hill, 1963.

Bedi J.S., Dawood M.Z. & Iqbal R., "Simulation of comma free synchronization scheme and expected error span in variable length codes", *Proc - 34th Midwest Symp. on Circuits & Systems*, Vol. 1, pp. 253-256, 1992.

Belongie M. & Heegard C., "Variable length trellis decoding", *IEEE ISIT '93*, San Antonio, Texas, USA, p. 261, 17-22 Jan. 1993.

Bernard M.A. & Sharma B.D., "Some combinatorial results on variable-length error-correcting codes", *ARS Combinatoria*, Vol. 25B, pp. 181-194, 1988.

Bernard M.A. & Sharma B.D., "A lower bound on average codeword length of variable length error-correcting codes", *IEEE Trans. Inform. Theory*, Vol. 36, No. 6, pp. 1474-1475, Nov. 1990.

Bernard M.A. & Sharma B.D., "Variable length perfect codes", *J. Inform. & Optimization Sciences*, Vol. 13, No. 1, pp. 143-151, 1992.

Buttigieg V., "Combined source and error-control coding", *M.Sc. Dissertation*, University of Manchester, 1992.

Buttigieg V. & Farrell P.G., "A maximum likelihood decoding algorithm for variable-length error-correcting codes", *Proc. 5th Bangor Symposium on Communications*, Bangor, Wales, pp. 56-59, 2-3 Jun. 1993a.

Buttigieg V., "Two algorithms to calculate the distance spectrum of VLEC codes", *Internal Report*, Comms. Res. Group, University of Manchester, pp. 1-15, Apr. 1994a.

Buttigieg V. & Farrell P.G., "On variable-length error-correcting codes", *Proc. 1994 IEEE ISIT*, Trondheim, Norway, p. 507, 27 Jun. - 1 Jul. 1994b.

Buttigieg V. & Farrell P.G., "Sequential decoding of variable-length error-correcting codes", *Proc. Eurocode 94*, Côte d'Or, France, pp. 93-98, 24-28 Oct. 1994c.

Buttigieg V. & Farrell P.G., "A maximum a-posteriori (MAP) decoding algorithm for variable-length error-correcting codes", *Codes and cyphers: Cryptography and coding IV*, Essex, England, The Institute of Mathematics and its Applications, pp. 103-119, 1995.

Calabi L. & Hartnett W.E., "A family of codes for the correction of substitution and synchronization errors", *IEEE Trans. Inform. Theory*, Vol. IT-15, No. 1, pp. 102-106, Jan. 1969a.

Calabi L. & Hartnett W.E., "Some general results of coding theory with applications to the study of codes for the correction of synchronization errors", *Inform. & Control*, Vol. 15, pp. 235-249, 1969b.

Calabi L. & Arquette L.K., "Basic properties of error-correcting codes", *Foundations of coding theory*, Dordrecht, Holland, D. Reidling Publishing Co., pp. 17-38, 1974a.

Calabi L. & Arquette L.K., "A study of error-correcting codes, I", *Foundations of coding theory*, Dordrecht, Holland, D. Reidling Publishing Co., pp. 40-59, 1974b.

Capocelli R.M., "A note on uniquely decipherable codes", *IEEE Trans. Inform. Theory*, Vol. IT-25, No. 1, pp. 90-94, Jan. 1979.

Capocelli R.M., "A decision procedure for finite decipherability and synchronizability of multivalued encodings", *IEEE Trans. Inform. Theory*, Vol. IT-28, No. 2, pp. 307-318, Mar. 1982.

Capocelli R.M., Gargano L. & Vaccaro U., "On the characterization of statistically synchronizable variable-length codes", *IEEE Trans. Inform. Theory*, Vol. 34, No. 4, pp. 817-825, Jul. 1988.

Capocelli R.M., De Santis A.A., Gargano L. & Vaccaro U., "On the construction of statistically synchronizable codes", *IEEE Trans. Inform. Theory*, Vol. 38, No. 2, pp. 407-14, Mar. 1992.

Cedervall M. & Johannesson R., "A fast algorithm for computing distance spectrum of convolutional codes", *IEEE Trans. Inform. Theory*, Vol. 35, No. 6, pp. 1146-1159, Nov. 1989.

Chevillat P.R. & Costello D.J., "Distance and computation in sequential decoding", *IEEE Trans. Commun.*, Vol. COM-24, pp. 440-447, Apr. 1976.

Chevillat P.R. & Costello D.J. Jr., "An analysis of sequential decoding for specific time-invariant convolutional codes", *IEEE Trans. Inform. Theory*, Vol. IT-24, No. 4, pp. 443-451, Jul. 1978.

Clark G.C. Jr & Cain J.B., *Error correction coding for digital communications*, New York, Plenum Press, 1981.

Dunscombe E.J., "Some applications of mathematics to coding theory", *Ph.D. Thesis*, Royal Holloway and Bedford New College, (University of London), 1988.

El Gamal A.A., Hemachandra L.A., Shperling I. & Wei V.K., "Using simulated annealing to design good codes", *IEEE Trans. Inform. Theory*, Vol. IT-33, No. 1, pp. 116-123, Jan. 1987.

Escott A., "A new performance measure for a class of two-length error correcting codes", *Codes and cyphers: Cryptography and Coding IV*, Essex, England, The Institute of Mathematics and its Applications, pp. 167-181, 1995.

Even S., "Tests for unique decipherability", *IEEE Trans. Inform. Theory*, Vol. IT-9, pp. 109-112, Apr. 1963.

Fano R.M., "A heuristic discussion of probabilistic decoding", *IEEE Trans. Inform. Theory*, Vol. IT-9, pp. 64-74, Apr. 1963.

Farrell P.G., "Linear binary anticodes", *Electron. Lett.*, Vol. 6, No. 13, pp. 419-421, Jun. 1970.

Farrell P.G. & Farrag A., "Further properties of linear binary anticodes", *Electron. Lett.*, Vol. 10, No. 16, p. 340, Aug. 1974.

Farrell P.G., "An introduction to anticodes", *Internal Report*, Kent, England, The University of Kent at Canterbury, 1977.

Ferguson T.J. & Rabinowitz J.H., "Self-synchronizing Huffman codes", *IEEE Trans. Inform. Theory*, Vol. IT-30, No. 4, pp. 687-693, Jul. 1984.

Forney G.D. Jr., "Convolutional codes III. Sequential decoding", *Inform. & Control*, Vol. 25, No. 3, pp. 267-297, Jul. 1974.

Gilbert E.N. & Moore E.F., "Variable-length binary encodings", *Bell Sys. Tech. J.*, Vol. 38, pp. 933-967, Jul. 1959.

Gilbert E.N., "Synchronization of binary messages", *IRE Trans. Inform. Theory*, Vol. IT-6, pp. 470-477, 1960.

Golomb S.W., Gordon B. & Welch L.R., "Comma-free codes", *Can. J. Math.*, Vol. 10, pp. 202-209, 1958.

Hartnett W.E. (Ed), *Foundations of coding theory*, Dordrecht, Holland, D. Reidling Publishing Co., 1974.

Hatcher T.R., "On a family of error-correcting and synchronizable codes", *IEEE Trans. Inform. Theory*, pp. 620-624, Sep. 1969.

Hazeltine B., "Regular expressions and variable length encodings", *IEEE Trans. Inform. Theory*, Vol. IT-9, p. 48, Jan. 1963.

Hollmann H.D.L., "A relation between Levenshtein-type distances and insertion-and-deletion correcting capabilities of codes", *IEEE Trans. Inform. Theory*, Vol. 39, No. 4, pp. 1424-1427, Jul. 1993.

Huffman D.A., "A method for the construction of minimum redundancy codes", *Proc. IRE*, Vol. 40, pp. 1098-1101, Sep. 1952.

Jacobs I.M. & Berlekamp E.R., "A lower bound to the distribution of computation for sequential decoding", *IEEE Trans. Inform. Theory*, Vol. IT-13, No. 2, pp. 167-174, Apr. 1967.

Jelinek F., "A fast sequential decoding algorithm using a stack", *IBM J. Res. and Dev.*, Vol. 13, pp. 675-685, Nov. 1969.

Jeruchim M.C., Balaban P. & Shannugan K.S., *Simulation of communications systems*, New York, Plenum Press, 1992.

Kendall W.B. & Reed I.S., "Path-invariant comma-free codes", *IRE Trans. Inform. Theory*, Vol. IT-8, pp. 350-355, Oct. 1962.

Kraft L.G., "A device for quantizing, grouping, and coding amplitude modulated pulses", *M.S. Thesis*, Electrical Engineering Department, M.I.T., Mar. 1949.

Kruskal J.B., "An overview of sequence comparison: Time warps, string edits, and macromolecules", *SIAM Review*, Vol. 25, No. 2, pp. 201-237, Apr. 1983.

Levenshtein V.I., "Binary codes with correction of deletions, insertions and substitution of symbols", *Dokl. Akad. Nank. SSSR*, Vol. 163, pp. 845-848, 1965a.

Levenshtein V.I., "Binary codes capable of correcting spurious insertions and deletions of ones", *Probl. Peredachi Inform.*, Vol. 1, No. 1, pp. 12-25, 1965b.

Levenshtein V.I., "Binary codes capable of correcting deletions, insertions and reversals", *Sov. Phys. Doklady*, Vol. 10, p. 707, 1966.

Levy J.E., "Self-synchronizing codes derived from binary cyclic codes", *IEEE Trans. Inform. Theory*, Vol. IT-12, No. 3, pp. 286-290, Jul. 1966.

Lin S. & Costello D.J., *Error control coding: Fundamentals and applications*, Englewood Cliffs, Prentice-Hall, 1983.

MacWilliams F.J. & Sloane N.J.A., *The theory of error correcting codes*, Amsterdam, North-Holland Publishing Comp., 1978.

Masek W.J. & Paterson M.S., "How to compute string-edit distances quickly", *Time-warps, string edits and macromolecules: Theory and practice of sequence comparison*, Reading, Massachusetts, Addison-Wesley, pp. 337-349, 1983.

Massey J.L., "Variable-length codes and the Fano metric", *IEEE Trans. Inform. Theory*, Vol. IT-18, No. 1, pp. 196-198, Jan. 1972.

Massey J.L., "Joint source and channel coding", *Communication Systems and Random Process Theory*, Alpen aan den Rijn, The Netherlands, Sijthoff & Noordhoff, Vol. NATO-ASI series E, No. 25, pp. 279-293, 1978.

Maxted J.C. & Robinson J.P., "Error recovery for variable length codes", *IEEE Trans. Inform. Theory*, Vol. IT-31, No. 6, pp. 794-801, Nov. 1985.

McEliece R.J., "The Viterbi decoding complexity of linear block codes", *Proc. 1994 IEEE ISIT*, Trondheim, Norway, p. 341, 27 Jun. - 1 Jul. 1994.

McMillan B., "Two inequalities implied by unique decipherability", *IRE Trans. Inform. Theory*, Vol. IT-2, pp. 115-116, Dec. 1956.

Monaco M.E. & Lawler J.M., "Corrections and additions to 'Error recovery for variable length codes'", *IEEE Trans. Inform. Theory*, Vol. IT-33, No. 3, pp. 454-456, May 1987.

Montgomery B.L. & Abrahams J., "Synchronization of binary source codes", *IEEE Trans. Inform. Theory*, Vol. IT-32, No. 6, pp. 849-854, Nov. 1986.

Neumann P.G., "Efficient error-limiting variable-length codes", *IRE Trans. Inform. Theory*, Vol. IT-8, pp. 292-304, Jul. 1962a.

Neumann P.G., "On a class of efficient error-limiting variable-length codes", *IRE Trans. Inform. Theory*, Vol. IT-8, No. 5, pp. S260-S266, Sep. 1962b.

Neumann P.G., "Error-limiting coding using information-lossless sequential machines", *IEEE Trans. Inform. Theory*, pp. 108-115, Apr. 1964.

Papoulis A., *Probability, random variables, and stochastic processes*, New York, McGraw-Hill, 1965.

Piret P., "Comma free error correcting codes of variable length, generated by finite-state encoders", *IEEE Trans. Inform. Theory*, Vol. IT-28, No. 5, pp. 764-775, Sep. 1982.

Pless V., "Remarks on greedy codes", *Lecture Notes in Comput. Sci.*, Vol. 573, pp. 58-67, 1992.

Pollara F., McEliece R.J. & Abdel-Ghaffar K., "Finite-state codes", *IEEE Trans. Inform. Theory*, Vol. 34, No. 5, pp. 1083-1089, Sep. 1988.

Rahman M. & Misbahuddin S., "Effect of a binary symmetric channel on the synchronisation recovery of variable length codes", *The Computer J.*, Vol. 32, No. 3, pp. 246-251, 1989.

Ramamoorthy C.V. & Tufts D.W., "Reinforced prefixed comma-free codes", *IEEE Trans. Inform. Theory*, Vol. IT-13, No. 3, pp. 366-371, Jul. 1967.

Rouanne M. & Costello D.J., "An algorithm for computing the distance spectrum of trellis codes", *IEEE J. Selected Areas in Commun.*, Vol. 7, No. 6, pp. 929-940, Aug. 1989.

Sardinas A.A. & Patterson G.W., "A necessary and sufficient condition for unique decomposition of coded messages", *Conv. Rec. IRE*, Pt. 8, pp. 104-108, 1953.

Sato K., "A decision procedure for the unique decipherability of multivalued encodings", *IEEE Trans. Inform. Theory*, Vol. IT-25, No. 3, pp. 356-360, May 1979.

Savage J.E., "Sequential decoding - the computation problem", *Bell System Tech. J.*, Vol. 46, No. 1, pp. 149-175, Jan. 1966a.

Savage J.E., "The distribution of the sequential decoding computation time", *IEEE Trans. Inform. Theory*, Vol. IT-12, No. 2, pp. 143-147, Apr. 1966b.

Scholtz R.A., "Codes with synchronization capability", *IEEE Trans. Inform. Theory*, Vol. IT-12, No. 2, pp. 135-142, Apr. 1966.

Scholtz R.A., "Maximal and variable word-length comma-free codes", *IEEE Trans. Inform. Theory*, Vol. IT-15, No. 2, pp. 300-306, Mar. 1969.

Scholtz R.A., "Frame synchronization techniques", *IEEE Trans. Commun.*, Vol. COM-28, No. 8, pp. 1204-1212, Aug. 1980.

Schouhamer Immink K.A., "Runlength-limited sequences", *Proc. IEEE*, Vol. 78, No. 11, pp. 1745-1759, Nov. 1990.

Sellers F.F., "Bit loss and gain correction code", *IRE Trans. Inform. Theory*, Vol. IT-8, pp. 35-38, 1962.

Shannon C.E., "A mathematical theory of communication", *Bell System Tech. J.*, Vol. 27, pp. 623-656, 1948.

Sklar B., *Digital communications: Fundamentals and applications*, Englewood Cliffs, New Jersey, Prentice Hall, 1988.

Sodha J. & Tait D., "Node synchronisation for high rate convolutional codes", *Electron. Lett.*, Vol. 28, No. 9, pp. 810-812, Apr. 1992.

Stiffler J.J., *Theory of synchronous communication*, Englewood Cliffs, NJ, Prentice-Hall, 1971.

Sweeney P., *Error control coding - An introduction*, New York, Prentice Hall, 1991.

Takishima Y., Wada M. & Murakami H., "Error states and synchronization recovery for variable length codes", *IEEE Trans. Commun.*, Vol. 42, No. 2/3, pp. 783-792, Feb. 1994.

Tanaka E. & Kasai T., "Synchronization and substitution error-correcting codes for the Levenshtein metric", *IEEE Trans. Inform. Theory*, Vol. IT-22, No. 2, pp. 156-162, Mar. 1976.

Tanenbaum A.S., *Computer Networks (2nd Ed)*, Englewood Cliffs, Prentice-Hall, 1988.

Tavares S.E. & Fukada M., "Matrix approach to synchronization recovery for binary cyclic codes", *IEEE Trans. Inform. Theory*, Vol. IT-15, No. 1, pp. 93-101, Jan. 1969.

Titchener M.R., "Digital encoding by means of new T-codes to provide improved data synchronisation and message integrity", *IEE Proc.*, Vol. 131, Pt. E, No. 4, pp. 151-153, Jul. 1984.

Titchener M.R., "A question of ambiguity: Introduction to the T-codes", *Personal communication*, Aug. 1988.

Ullman J.D., "Near-optimal, single-synchronization-error-correcting code", *IEEE Trans. Inform. Theory*, Vol. IT-12, No. 4, pp. 418-424, Oct. 1966.

Ullman J.D., "On the capabilities of codes to correct synchronization errors", *IEEE Trans. Inform. Theory*, Vol. IT-13, No. 1, pp. 95-105, Jan. 1967.

Vembu S., Verdú S. & Steinberg Y., "The source-channel separation theorem revisited", *IEEE Trans. Inform. Theory*, Vol. 41, No. 1, pp. 44-54, Jan. 1995.

Viterbi A.J., "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm", *IEEE Trans. Inform. Theory*, Vol. IT-13, pp. 260-269, 1967.

Viterbi A.J., "Convolutional codes and their performance in communication systems", *IEEE Trans. Inform. Theory*, Vol. COM-19, No. 5, pp. 751-772, Oct. 1971.

Viterbi A.J. & Omura J.K., *Principles of digital communication and coding*, New York, McGraw-Hill, 1979.

Wei V.K. & Scholtz R.A., "On the characterization of statistically synchronizable codes", *IEEE Trans. Inform. Theory*, Vol. IT-26, No. 6, pp. 733-735, Nov. 1980.

Wolf J.K., "Efficient maximum likelihood decoding of linear block codes using a trellis", *IEEE Trans. Inform. Theory*, Vol. IT-24, No. 1, pp. 76-80, Jan. 1978.

Zigangirov K.S., "Some sequential decoding procedures", *Probl. Peredach. Inform.*, Vol. 2, No 2, pp. 13-25, 1966.

Appendix A

VLEC Codes for the 26-Symbol English Source and the 128-Symbol ASCII Source

Source Symbol	Probability	α_1-prompt Code	$\alpha_{1,1}$-prompt Code
e	0.1270	0000000	0000000
t	0.0906	1000110	0001111
a	0.0817	0100101	0010011
o	0.0751	0010011	0011100
i	0.0697	0001111	0100101
n	0.0674	1100011	0101010
s	0.0633	1010101	0110110000
h	0.0609	1001001	0110110111
r	0.0599	0110110	0111001000
d	0.0425	0101010	0111001111
l	0.0403	0011100	1000110000
c	0.0278	1110000	1000110111
u	0.0276	1101100	1001001000
m	0.0241	1011010000	1001001111
w	0.0236	1011010111	1010101000
f	0.0223	0111001000	1010101111
g	0.0202	0111001111	1011010000
y	0.0197	111111111100	1011010111
p	0.0193	111111100111	1100011000
b	0.0149	1111111100011	1100011111
v	0.0098	1111111010101	1101100000
k	0.0077	11111110000000	1101100111
j	0.0015	11111111101100	1110000000
x	0.0015	1111111001001000	1110000111
q	0.0010	1111110100101000	1111111000
z	0.0007	1111110100101111	1111111111

Table A.1: α_1-prompt and $\alpha_{1,1}$-prompt codes for the 26-symbol English source.

Source Symbol	C_{15} R =0.5894	C_{16} $R = 0.5890$	C_{17} $R = 0.5894$	C_{18} $R = 0.5228$	C_{19} $R = 0.5584$
e	011111	011100	11110	000000001	0011000
t	0000111	0001101	010111	000111111	10000000
a	0011000	0010010	0011101	011000110	01100011
o	10111011	01001000	01101001	101011000	000010010
i	01001001	10010001	10011000	110101010	011001000
n	10000100	10101110	00000110	111110101	010100101
s	101011010	100000100	110010100	1111111100	111110011
h	011000110	101110001	000100010	0101001110	100001111
r	101101101	111001111	001011000	1001100010	101111100
d	110010101	010111010	0010000010	1000001101	1111001001
l	1010001001	1001011010	1010111011	0010101000	1010101100
c	1110010110	0100111110	0100001111	1010010110	0010011011
u	0101001110	1010100100	1000010100	1111000001	0001101111
m	1001110111	1100010101	01000000000	0011011011	0110100010
w	1101100000	01000110001	10000011011	0100010000	1101111010
f	10101000100	01101111010	10101101100	0110100111	0111110101
g	01010010011	10110111000	100000111000	1100111011	1001010100
y	11100110001	11001011111	100011110101	10101111111	10100101010
p	11000111110	10001101100	0010010111001	11000001110	11110111100
b	111000001101	10011100011	1000111110100	11100110100	10101010111
v	100000111010	110010110000	00100101111101	01001010101	11010001011
k	100100100001	110001001011	10000001110100	10010010111	01111010001
j	110100010100	110101100110	10001101111011	00000110010	11000110001
x	1110100001100	110111111101	10101111001010	01111100110	00010110010
q	1111001000000	111111000000	101011110011011	01011011010	00101101101
z	1101010100010	1101010011011	100011011111001	11101001001	01001100110

Table A.2: Various VLEC codes for the 26-symbol English source with $d_{\text{free}} = 5$

Source Symbol	C_{20}, d_{free}=5 R =0.5906	C_{21}, d_{free}=7 R=0.4673
e	011100	1111111
t	0010010	01110001
a	00111110	010010111
o	10010001	0010110100
i	01000111	0001001001
n	11101000	00011000101
s	001010110	10100011000
h	110000100	100110010001
r	101101111	001000101011
d	000110001	010011000110
l	110011011	101001011100
c	1010001010	0001110111001
u	0100101011	1010100001101
m	1101010011	1000011010000
w	1000111101	1100010101110
f	0101110100	00010111101001
g	01001001010	00101000110110
y	11100110111	10001010001101
p	10111101011	11000000111000
b	11011011111	01000110000010
v	11101111000	000101111001101
k	10010110000	101010011111000
j	100001101100	100101010010000
x	100010111010	010001100011011
q	010011000110	100011000101101
z	110110101001	110000000000110

Table A.3: Two VLEC codes constructed using the heuristic construction with the majority voting algorithm for the 26-symbol English source

Source Symbol	Probability	C_{22}, d_{free}=5, R=0.7292	C_{23}, d_{free}=7, R=0.4632
space	1.09×10^{-1}	00000000	000000000000010
ht	9.58×10^{-2}	00011111	100001110110110
e	7.20×10^{-2}	11100011	010001101101000
t	5.16×10^{-2}	11111100	110000011011100
o	5.15×10^{-2}	000100101	001011011100100
i	4.68×10^{-2}	111001101	101010101010000
nl	4.30×10^{-2}	010010011	011010110001110
n	4.15×10^{-2}	100111000	111011000111010
s	3.82×10^{-2}	0000101010	000110111111010
r	3.77×10^{-2}	0001000111	100111001001110
a	3.32×10^{-2}	0010010100	010111010010000
d	2.60×10^{-2}	0101011001	110110100100100
c	2.53×10^{-2}	1010001001	001101100011100
h	2.33×10^{-2}	1100110011	101100010101000

Source Symbol	Probability	C_{22}, d_{free}=5, R=0.7292	C_{23}, d_{free}=7, R=0.4632
l	2.17×10^{-2}	1101100100	011100001110110
f	1.95×10^{-2}	00000110110	111101111000010
*	1.91×10^{-2}	00001101001	000010000111101
m	1.80×10^{-2}	00101001110	100011110001001
u	1.77×10^{-2}	00110110001	010011101010111
b	1.37×10^{-2}	01010101111	110010011100011
p	1.23×10^{-2}	01011000100	001001011011011
g	1.15×10^{-2}	01110011010	101000101101111
N	1.12×10^{-2}	10110000111	011000110110001
_	7.72×10^{-3}	11001010001	111001000000101
.	7.26×10^{-3}	000001011010	000100111000101
w	7.24×10^{-3}	000110001011	100101001110001
y	6.47×10^{-3}	000110100100	010101010101111
,	6.41×10^{-3}	001001110111	110100100011011
(	6.33×10^{-3}	001111010000	001111100100011
\	6.31×10^{-3}	011001001100	101110010010111
)	6.26×10^{-3}	011010111010	011110001001001
-	5.98×10^{-3}	101101101010	111111111111101
/	5.88×10^{-3}	110001110000	0000100110101101
[	5.75×10^{-3}	110111010110	1000111011000101
;	5.62×10^{-3}	111110110101	0100111101111001
]	5.57×10^{-3}	0001001101110	1100100000010001
v	5.46×10^{-3}	0001010110000	0010010001100001
=	4.70×10^{-3}	0010001111011	1010001100001001
0	4.44×10^{-3}	0010010001110	0110001010110101
1	4.03×10^{-3}	0010110101001	1110010111011101
I	3.65×10^{-3}	0011101010010	0001001001011101
T	3.16×10^{-3}	0101101010101	1001010100110101
#	3.11×10^{-3}	0101110000110	0101010010001001
E	3.11×10^{-3}	0110110110010	1101001111100001
C	2.29×10^{-3}	1000011001000	0011111110010001
k	2.28×10^{-3}	1000111010111	1011100011111001
V	2.16×10^{-3}	1001101001011	0111100101000101
S	1.87×10^{-3}	1010100100110	1111111000101101
2	1.77×10^{-3}	1101001110010	0000000111010011
'	1.70×10^{-3}	00000101010100	1000011010111011
:	1.67×10^{-3}	00001011001010	0100011100000111
W	1.64×10^{-3}	00010101101111	1100000001101111
A	1.61×10^{-3}	00100110101110	0010110000011111
O	1.61×10^{-3}	00111001010110	1010101101110111
x	1.57×10^{-3}	00111001101000	0110101011001011
"	1.56×10^{-3}	01001110010101	1110110110100011
L	1.56×10^{-3}	01011010000110	0001101000100011

Source Symbol	Probability	C_{22}, d_{free}=5, R=0.7292	C_{23}, d_{free}=7, R=0.4632
D	1.52×10^{-3}	01100101110111	1001110101001011
R	1.44×10^{-3}	01110010111001	0101110011110111
3	1.39×10^{-3}	01110111001010	1101101110011111
+	1.36×10^{-3}	10001110110110	0011011111101111
q	1.26×10^{-3}	10101001111011	1011000010000111
9	1.15×10^{-3}	11010101110010	0111000100111011
M	1.05×10^{-3}	11011101101001	1111011001010011
<	9.51×10^{-4}	000011001001011	00001010100110010
B	9.18×10^{-4}	000011100101110	10001101111100010
z	8.52×10^{-4}	000101001011000	01001100010011010
4	7.54×10^{-4}	000110010100110	11001011001001010
U	7.05×10^{-4}	001001101010110	00100111010101010
>	6.56×10^{-4}	001010101110101	10100000001111010
Y	6.56×10^{-4}	001010110111011	01100001100000010
}	6.39×10^{-4}	001111010110000	11100110111010010
6	6.23×10^{-4}	001111011001110	00010001011010010
&	6.06×10^{-4}	010011111000110	10010110000000010
{	5.90×10^{-4}	010100011001010	01010111101111010
P	5.74×10^{-4}	011110101101111	11010000110101010
5	5.57×10^{-4}	011111101011010	00111100101001010
F	4.42×10^{-4}	100000101110010	10111011110011010
8	4.26×10^{-4}	101011110110110	01111010011100010
7	3.93×10^{-4}	110010111011010	11111101000110010
$\vert$	3.77×10^{-4}	110111110101010	00000010111001110
%	3.77×10^{-4}	111100101010100	10000101100011110
H	3.77×10^{-4}	0000010101110110	01000100001100110
!	2.95×10^{-4}	0000110011001000	11000011010110110
G	2.29×10^{-4}	0000111001101111	00101111001010110
j	1.80×10^{-4}	0001100101010010	10101000010000110
K	1.64×10^{-4}	0010011010100110	01101001111111110
Q	1.47×10^{-4}	0010111111011011	11101110100101110
X	1.47×10^{-4}	0011001100101110	00011001000101110
?	3.28×10^{-5}	0011011001010110	10011110011111110
Z	1.64×10^{-5}	0011100010001110	01011111110000110
$\$$	1.64×10^{-5}	0100011111010111	11011000101010110
@	1.64×10^{-5}	0101111011011010	00110100110110110
^	1.64×10^{-5}	0110000101110101	10110011101100110
`	1.64×10^{-5}	0111110110111011	01110010000011110
~	1.64×10^{-5}	1001101110110110	11110101011001110
ack	1.64×10^{-5}	1010110110001010	00000011001100001
bel	1.64×10^{-5}	1011011011001010	10000100010110001
bs	1.64×10^{-5}	1100101001001011	01000101111001001
can	1.64×10^{-5}	1101111100101110	11000010100011001

Source Symbol	Probability	C_{22}, d_{free}=5, R=0.7292	C_{23}, d_{free}=7, R=0.4632
cr	1.64×10^{-5}	1111011001101111	0010111011111001
dc2	1.64×10^{-5}	00000011110010110	1010100110010 1001
dc3	1.64×10^{-5}	0000101111101 1011	0110100000101000 1
dc4	1.64×10^{-5}	0010011110100101 1	1110111101000000 1
dcl	1.64×10^{-5}	00101111001101110	0001100011000000 1
del	1.64×10^{-5}	00110100101010010	1001111110101000 1
dle	1.64×10^{-5}	00110111010110110	0101111000010100 1
em	1.64×10^{-5}	00111011110001110	1101100101111100 1
enq	1.64×10^{-5}	0011111010111010 1	0011010100001100 1
eot	1.64×10^{-5}	01000111011011010	1011001001100100 1
esc	1.64×10^{-5}	01001111110101010	0111001111011000 1
etb	1.64×10^{-5}	01101100010001110	1111010010110000 1
etx	1.64×10^{-5}	01111011011010110	0000101101001110 1
fs	1.64×10^{-5}	01111101001011000	1000110000100110 1
gs	1.64×10^{-5}	1000111100111101 1	0100110110011010 1
J	1.64×10^{-5}	10010111001001010	1100101011110010 1
nak	1.64×10^{-5}	10011010101010100	0010011010000010 1
np	1.64×10^{-5}	10011110110101110	1010000111101010 1
nul	1.64×10^{-5}	1101001010111011 1	0110000001010110 1
rs	1.64×10^{-5}	11110110011001000	1110011100111110 1
si	1.64×10^{-5}	1111011110110111 1	0001000010111110 1
so	1.64×10^{-5}	1111111111101101 1	1001011111010110 1
soh	1.64×10^{-5}	000001111110100110	0101011001101010 1
stx	1.64×10^{-5}	00001011010111011 1	1101000100000010 1
sub	1.64×10^{-5}	001011011110010110	0011110101110010 1
syn	1.64×10^{-5}	001011110011011010	1011101000011010 1
us	1.64×10^{-5}	00110011111101101 1	0111101110100110 1
vt	1.64×10^{-5}	001101111001001010	1111110011001110 1

Table A.4: Two VLEC codes for the 128-symbol ASCII source derived from a *C*-program

Appendix B

Two Algorithms to Calculate the Distance Spectrum of VLEC Codes

Internal Report, Comms. Res. Group, University of Manchester, pp. 1-15, Apr. 1994.

Appendix C

Published Papers

"A maximum likelihood decoding algorithm for variable-length error-correcting codes",

Proc. 5th Bangor Symposium on Communications,

Bangor, Wales,

pp. 56-59,

2-3 Jun. 1993.

"On variable-length error-correcting codes",

Proc. 1994 IEEE ISIT,

Trondheim, Norway,

pp. 507,

27 Jun. - 1 Jul. 1994.

“Sequential decoding of variable-length error-correcting codes”,

Eurocode 94,

Côte d'Or, France,

pp. 93-98,

24-28 Oct. 1994.

"A maximum a-posteriori (MAP) decoding algorithm for variable-length error-correcting codes",

Codes and cyphers: Cryptography and coding IV,

Essex, England,

The Institute of Mathematics and its Applications,

pp. 103-119, 1995.

Index